GemStone®

System Administration Guide for GemStone/S 64 Bit

Version 3.0

June 2011

vmware®

GEMSTONE S™ 64

INTELLECTUAL PROPERTY OWNERSHIP

COPYRIGHTS

PATENTS

GemStone software is covered by U.S. Patent Number 6,256,637 "Transactional virtual machine architecture", Patent Number 6,360,219 "Object queues with concurrent updating", Patent Number 6,567,905 "Generational garbage collector with persistent object cache", and Patent Number 6,681,226 "Selective pessimistic locking for a concurrently updateable database". GemStone software may also be covered by one or more pending United States patent applications.

TRADEMARKS

VMware, Inc.
15220 NW Greenbrier Parkway
Suite 150
Beaverton, OR 97006

Preface

About This Manual

This manual tells how to perform day-to-day administration of your GemStone/S 64 Bit repository.

Prerequisites

This manual is intended for users that are at least somewhat familiar with using Smalltalk and the Topaz programming environment to execute GemStone Smalltalk code. It also assumes some familiarity with UNIX.

You should have the GemStone system installed correctly on your host computer, as described in the *GemStone/S 64 Bit Installation Guide* for your platform.

How This Manual is Organized

This manual is organized in three parts: initial configuration, day-to-day administration, and appendixes.

Part 1: System Configuration

- Chapter 1, "Configuring the GemStone Server," tells how to adapt the GemStone central repository server to the needs of your application. Three sample configuration files are provided as starting points.
- Chapter 2, "Configuring Gem Session Processes," tells how to configure the GemStone processes that provide the services to individual application clients.
- Chapter 3, "Connecting Distributed Systems," explains the additional steps necessary to run GemStone in a networked environment. It includes examples of how to set up common configurations.

Part 2: System Administration

- Chapter 4, "Running GemStone," tells how to start and stop the GemStone system, how to troubleshoot startup problems, how to deal with unexpected shutdowns, and how to bulk-load objects.
- Chapter 5, "Monitoring GemStone," explains where the system logs are located, how to audit the repository, and how to monitor the performance of the GemStone server and its clients using GemStone Smalltalk methods.
- Chapter 6, "User Accounts and Security," introduces the tools available for administration tasks and details how to log in to the repository, and how to create, modify, and remove GemStone user accounts. It also tells how to configure GemStone login security by restricting valid passwords, imposing password and account age limits, and monitoring intrusion attempts.
- Chapter 7, "Managing Repository Space," gives procedures for managing the repository itself: checking free space, adding space, and controlling its growth. It also how to recover from disk-full conditions.
- Chapter 8, "Managing Transaction Logs," gives procedures for setting up the optional full incremental logging, managing log space, and archiving the log files.
- Chapter 9, "Making and Restoring Backups," gives procedures for making a GemStone full backup while the repository is in use, and for using backups and transaction logs to restore the repository.
- Chapter 10, "Managing Memory," discusses process memory, how to configure it, and how to diagnose problems.
- Chapter 11, "Managing Growth," presents the main concepts underlying garbage collection in GemStone and tells when, how, and why to invoke the garbage collection mechanisms.

Part 3: Appendixes

- Appendix A, "GemStone Configuration Options," explains how GemStone uses configuration files and describes each configuration option.
- Appendix B, "GemStone Utility Commands," describes each of the GemStone-supplied commands defined for use by the GemStone system administrator.
- Appendix C, "Network Resource String Syntax," lists the syntax for network resource strings, which allow you to specify the host machine for a GemStone file or process.
- Appendix D, "GemStone Kernel Objects," lists the GemStone-supplied objects that are present in your repository after the GemStone system has been successfully installed.
- Appendix E, "Environment Variables," lists all environment variables used by GemStone, including those that are reserved.
- Appendix F, "Localization," explains how to use Locale to handle non-US decimal points, and Extended Character Sets to allow sort and collation on Characters beyond the ASCII range.
- Appendix G, "statmonitor and VSD Reference," describes how to use the performance-tuning tools statmonitor and VSD.
- Appendix H, "Cache Statistics," lists and defines all the cache statistics that are available to monitor GemStone.
- Appendix I, "Object State Change Tracking," describes how to analyze tranlogs to track the history of object modifications.

Terminology Conventions

The term "GemStone" is used to refer to the server products GemStone/S 64 Bit and GemStone/S; the GemStone Smalltalk programming language; and may also be used to refer to the company, previously GemStone Systems, Inc., now a division of VMware, Inc.

Typographical Conventions

This document uses the following typographical conventions:

- Operating system and Topaz commands are shown in **bold** typeface.

- Smalltalk methods, GemStone environment variables, operating system file names and paths, listings, and prompts are shown in `monospace` typeface.
- Interactive dialogue from GemStone is shown in an `underlined monospace` typeface.
- Lines you type are distinguished from system output by boldface type
- Place holders that are meant to be replaced with real values are shown in *italic* typeface.
- Literals are shown in **bold** typeface.
- Optional arguments and terms are enclosed in [square brackets].
- Braces { } mean 0 or more modifiers.
- Alternative arguments and terms are separated by a vertical bar (|).

Other GemStone Documentation

You may find it useful to look at documents that describe other GemStone system components:

- *Topaz Programming Environment* – describes Topaz, a scriptable command-line interface to GemStone Smalltalk. Topaz is most commonly used for performing repository maintenance operations.
- *Programming Guide for GemStone/S 64 Bit* – a programmer's guide to GemStone Smalltalk, GemStone's object-oriented programming language.
- *GemBuilder for Smalltalk Users's Guide* — describes GemBuilder for Smalltalk, a programming interface that provides a rich set of features for building and running client Smalltalk applications that interact transparently with GemStone Smalltalk.
- *GemBuilder for C* — describes GemBuilder for C, a set of C functions that provide a bridge between your application's C code and the application's database controlled by GemStone.

In addition, each release of GemStone/S 64 Bit includes *Release Notes*, describing changes in that release, and platform-specific *Installation Guides*, providing system requirements and installation and upgrade instructions.

A description of the behavior of each GemStone kernel class is available in the class comments in the GemStone Smalltalk repository. Method comments include a description of the behavior of methods.

Technical Support

GemStone Website

http://support.gemstone.com

GemStone's Technical Support website provides a variety of resources to help you use GemStone products:

- **Documentation** for released versions of all GemStone products, in PDF form.
- **Downloads** and **Patches**, including past and current versions of GemBuilder for Smalltalk.
- **Bugnotes**, identifying performance issues or error conditions you should be aware of.
- **TechTips**, providing information and instructions that are not otherwise included in the documentation.
- **Compatibility matrices**, listing supported platforms for GemStone product versions.

This material is updated regularly; we recommend checking this site on a regular basis.

Help Requests

You may need to contact Technical Support directly, if your questions are not answered in the documentation or by other material on the Technical Support site. Technical Support is available to customers with current support contracts.

Requests for technical support may be submitted online or by telephone. We recommend you use telephone contact only for serious requests that require immediate attention, such as a production system down. The support website is the preferred way to contact Technical Support.

Website: http://techsupport.gemstone.com

Email: techsupport@gemstone.com

Telephone: (800) 243-4772 or (503) 533-3503

When submitting a request, please include the following information:

- Your name, company name, and GemStone server license number.
- The versions of all related GemStone products, and of any other related products, such as client Smalltalk products.
- The operating system and version you are using.
- A description of the problem or request.
- Exact error message(s) received, if any, including log files if appropriate.

Technical Support is available from 8am to 5pm Pacific Time, Monday through Friday, excluding VMware/GemStone holidays.

24x7 Emergency Technical Support

GemStone Technical Support offers, at an additional charge, 24x7 emergency technical support. This support entitles customers to contact us 24 hours a day, 7 days a week, 365 days a year, if they encounter problems that cause their production application to go down, or that have the potential to bring their production application down. For more details, contact your GemStone account manager.

Training and Consulting

Consulting is available to help you succeed with GemStone products. Training for GemStone software is available at your location, and training courses are offered periodically at our offices in Beaverton, Oregon. Contact your GemStone account representative for more details or to obtain consulting services.

Contents

Chapter 8. Managing Transaction Logs 243

Chapter 9. Making and Restoring Backups 257

Appendix F. Localization *443*

Appendix G. statmonitor and VSD Reference *457*

Chapter

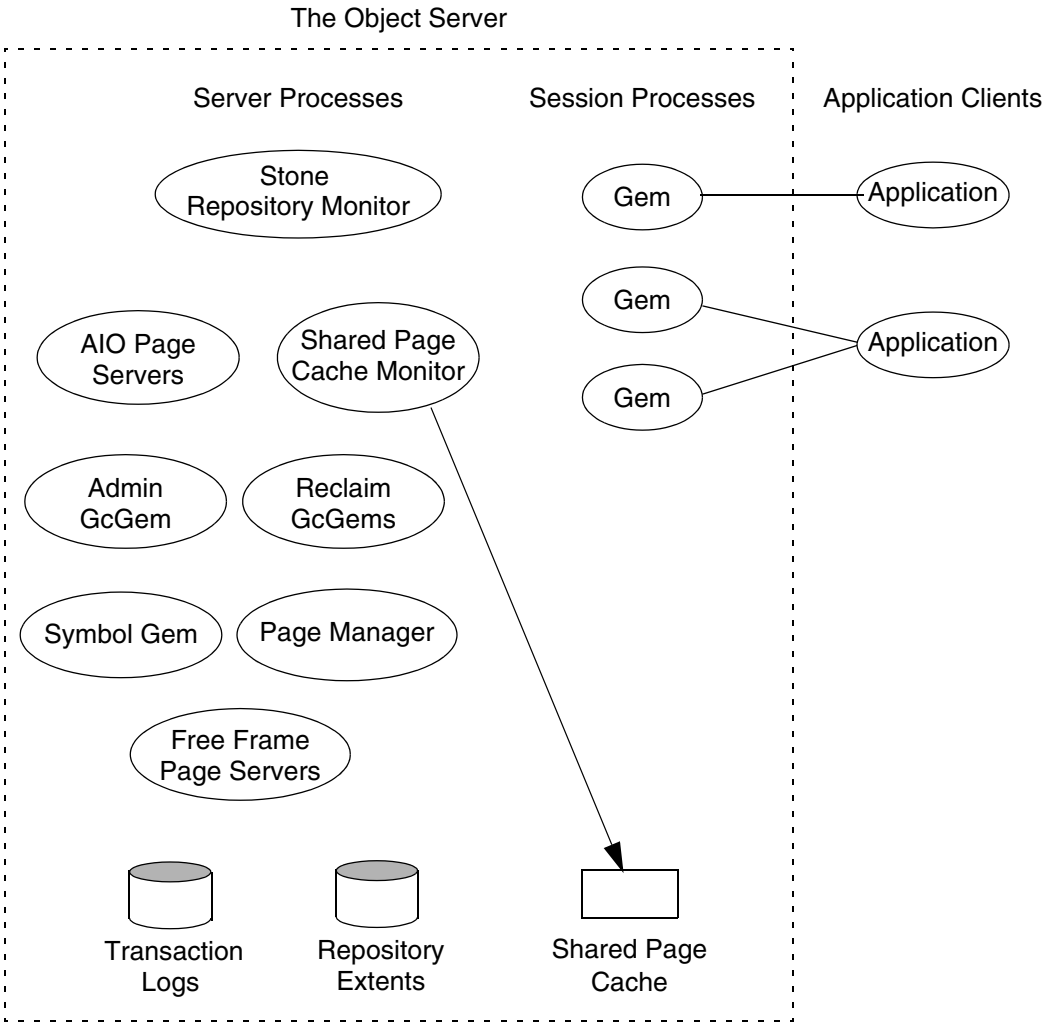
1 Configuring the GemStone Server

Figure 1.1 shows the basic GemStone/S 64 Bit architecture as seen by its administrator. The object server can be thought of as having two active parts. The *server processes* consist of the Stone repository monitor and a set of subordinate processes. These processes provide resources to individual Gem *session processes*, which are servers to application clients.

This chapter tells you how to configure the GemStone server processes, repository, transaction logs, and shared page cache to meet the needs of a specific application. For information about configuring session processes for clients, refer to Chapter 2.

The elements shown in Figure 1.1 can be distributed across multiple nodes to meet your application's needs. For information about establishing distributed servers, refer to Chapter 3.

Figure 1.1 The GemStone Object Server



1.1 Configuration Overview

Figure 1.1 shows the key parts that define the server configuration:

- The *Stone repository monitor* process acts as a resource coordinator. It synchronizes critical repository activities and ensures repository consistency.
- The *shared page cache monitor* creates and maintains a *shared page cache* for the GemStone server. The monitor balances page allocation among processes, ensuring that a few users or large objects do not monopolize the cache. The size of the shared page cache is configurable and should be scaled to match the size of the repository and the number of concurrent sessions; a larger size often provides better performance.
- The *AIO page servers* perform asynchronous I/O for the Stone repository monitor. Their primary tasks are to update the extents periodically and to pre-allocate (grow) the extents at startup when that feature is enabled. The default configuration uses one AIO page server, but additional ones should be specified for systems having several extents.
- The *Admin GcGem* is a Gem server process that is dedicated to performing the administrative garbage collection tasks under supervision of the Stone. Each repository can have up to one Admin GcGem process running.
- The *Reclaim GcGems* perform page reclaim operations on both shadow objects and dead objects. On a running GemStone system, there may be between 0 and *n* Reclaim GcGems present, where *n* is the number of extents in the repository.
- The *Symbol Gem* is a Gem server background process that is responsible for creating all new Symbols, based on session requests that are managed by the Stone.
- The *Page Manager* is a background process that assists the Stone with page disposal in coordination with the remote page caches.
- The *Free Frame Page Servers* are Gem server processes that are dedicated to the task of adding free frames to the free frame list, from which a Gem can take as needed. The default configuration uses one free frame page server, but you can configure as many as 30 free frame page server processes.
- Objects are stored on the disk in one or more *extents,* which can be files in the file system, data in raw partitions, or a mixture. The location of each extent is configurable.
- *Transaction logs* permit recovery of committed data if a system crash occurs. They also reduce disk activity by eliminating the need to flush to the extents all data pages written by each transaction. The optional *full logging mode* allows

transaction logs to be used with GemStone backups for full recovery of committed transactions in the event of media failure.

The transaction logs should reside on a different disk drive (spindle) from the extents, and neither should be on a drive that contains the operating system swap space (sometimes called page space).

The Server Configuration File

At start-up time, GemStone reads a system-wide configuration file. By default this file is `$GEMSTONE/data/system.conf`, where GEMSTONE is an environment variable that points to the directory in which the GemStone software is installed.

Appendix A, "GemStone Configuration Options," tells how to specify an alternate configuration file and how to use supplementary files to adjust the system-wide configuration for a specific GemStone executable. The appendix also describes each of the configuration options.

Here is a brief summary of important facts about the configuration file:

- Lines that begin with # are comments. Settings supplied as comments are the same as the default values. You can easily change the configuration by altering the option value and moving the # symbol to the line previously in force.
- Options that begin with "GEM_" are read only by Gem session processes at the time they start. Chapter 2, "Configuring Gem Session Processes," describes their use.
- Options that begin with "SHR_" are read both by the Stone repository monitor and by the first Gem session process to start on a node remote from the Stone. These options configure the local shared page cache.
- Most other options (those not beginning with "GEM_" or "SHR_") are read by the Stone repository monitor. If another GemStone process needs that information, it is exchanged through a TCP/IP connection with the Stone.
- If an option is defined more than once, only the last definition is used. Certain run-time configuration changes, such as the addition of an extent, cause the repository monitor to append new configuration statements to the file. Be sure to check the end of a configuration file for possible entries that override earlier ones.
- Most configuration parameters have default values. If you do not specify a setting for the parameter, or if there is an error in your settings such as an out of range value, the default is used. After making changes, check the log file headers to ensure that your configuration setting is correct.

Example Configuration Settings

This section describes possible sets of configuration parameter settings that you can use as a starting point for setting up a configuration for your particular system. Although these settings differ in a number of ways, the primary difference is in the size of application they accommodate.

Small	Handles I/O more efficiently than the initial configuration by using separate drives (spindles) for the extents and transaction logs. A larger page cache supports more users. Full transaction logging provides real-time incremental backup.
Medium	Uses raw disk partitions for possibly increased throughput. It accommodates more users and a larger repository.
Large	Uses multiple extents to accommodate a repository of 100 GB.

Table 1.1 shows an example of possible configuration changes for these three systems. The numbers would be adjusted for the resources and requirements of your particular system, then applied to the default configuration file, `$GEMSTONE/data/system.conf`.

NOTE

Large systems will almost certainly require additional tuning for optimal performance. Very large systems will probably need to be distributed across several servers. For large systems, consider consulting GemStone Professional Services for expert assistance in establishing your configuration.

More information about these settings is provided in the detailed instructions for establishing your own configuration, beginning on page 34.

Gemservers for Large Configurations

There is an upper limit to the number of processes that you can run before performance becomes unacceptable. For very large configurations, we recommend a separate Gem server machine with a remote shared page cache set up specifically to run Gem sessions, with a high bandwidth connection between the repository server and the Gem server.

Table 1.1 Settings for Selected Configurations

Characteristic or Configuration Option	Server Configuration		
	Small	Medium	Large
Application Characteristics			
Maximum number of user sessions	12	250	1500
Repository size	100 MB	10 GB	100 GB
System Requirements			
Typical number of CPUs	1–2	2-4	8+
RAM	512 MB	8000 MB	128000 MB
Kernel shared memory	26 MB	1600 MB	64500 MB
Number of disk drives	3	13	35
Configuration Settings			
STN_MAX_SESSIONS	40	280	1658
SHR_PAGE_CACHE_SIZE_KB	75000	1500000	64000000
STN_PRIVATE_PAGE_CACHE_KB	(default)	16384	32768
STN_CHECKPOINT_INTERVAL	300	600	900
STN_CR_BACKLOG_THRESHOLD	(default)	500	3000
STN_SIGNAL_ABORT_CR_BACKLOG	20	400	2800
STN_FREE_SPACE_THRESHOLD	10	100	100
STN_NUM_LOCAL_AIO_SERVERS	1	4	10
SHR_NUM_FREE_FRAME_SERVERS	1	2	5

Table 1.1 Settings for Selected Configurations (Continued)

Characteristic or Configuration Option	Server Configuration		
	Small	Medium	Large
Approximate Memory Usage			
Stone repository monitor (MB)	20	40	100
Each Gem session process[a] (MB)	20	20	25
Extents			
DBF_EXTENT_NAMES	(1 file)	(8 files)	(25 files)
DBF_EXTENT_SIZES	(Unlimited)	(1995 each)[b]	(3995 each)[b]
DBF_PRE_GROW	False	True	True
DBF_ALLOCATION_MODE	(Not used)	10,10,10,...	10,10,10,...
Transaction Logs			
STN_TRAN_FULL_LOGGING	True	True	True
STN_TRAN_LOG_DIRECTORIES	(2 directories)	(5 raw partitions)	(10 raw partitions)
STN_TRAN_LOG_SIZES	50,50	499[b] each	1995[b] each
Stone Response to Gem Fatal Errors			
STN_HALT_ON_FATAL_ERR	False[c]	False[c]	False[c]
Garbage Collection			
STN_NUM_GC_RECLAIM_SESSIONS	1	2	5
STN_ADMIN_GC_SESSION_ENABLED	True	True	True

[a] Depends on the value of GEM_TEMPOBJ_CACHE_SIZE (default=10 MB but a larger setting is often required).

[b] For best performance, set DBF_EXTENT_NAMES and STN_TRAN_LOG_SIZES to slightly less than the actual size of the partition. The values given for extents are based on 2 GB partitions in the Medium configuration and 4 GB partitions in the Large configuration.

[c] For deployed systems, a setting of False is recommended, to avoid unnecessary Stone shutdown. A setting of True provides more protection for the repository during testing and development.

Recommendations About Disk Usage

You can enhance server performance by distributing the repository files on multiple disk drives. Under certain circumstances and for certain operating systems, placing the data in raw disk partitions rather than in a file system can enhance performance.

Why Use Multiple Drives?

Efficient access to GemStone repository files requires that the server node have at least three disk drives (that is, three separate spindles or physical volumes) to reduce I/O contention. For instance:

- One spindle for swap space and the operating system (GemStone executables can also reside here).
- One spindle for the repository extent, perhaps with a lightly accessed file system sharing the drive.
- One spindle for transaction logs (with least two raw partitions or directories) and possibly user file systems if they are only lightly used for non-GemStone purposes.

When developing your own configuration, bear in mind the following guidelines:

1. Keep extents and transaction logs separate from operating system swap space. Don't place either extents or logs on any spindle that contains a swap partition; doing so drastically reduces performance.
2. Place the transaction logs on a spindle that does not contain extents. Placing logs on a different spindle from extents increases the transaction rate for updates while reducing the impact of updates on query performance. You can place multiple logs on the same spindle, since only one log file is active at a time.

NOTE

Under operating systems that use volume managers, you need to be aware of how logical volume groups are assigned to disk drives (physical volumes). You should try to assign each of the above (swap, extents, and transaction logs) to a different disk drive.

3. To benefit from multiple extents on multiple spindles, you must use weighted allocation mode. If you use sequential allocation, multiple extents provide no benefit. For details about weighted allocation, see "Allocating Data to Multiple Extents" on page 47.
4. In addition, if you decide to use more than one AIO page server, you'll need to keep extents on several different spindles. You'll derive no advantage from multiple page servers unless they can write different pages to different extents simultaneously, instead of contending for the same disk drive head.

When to Use Raw Partitions

Each raw partition (sometimes called a raw device or raw logical device) is like a single large sequential file, with one extent or one transaction log per partition. The use of raw disk partitions can yield better performance, depending on how they are used and the balancing of system resources:

- Placing transaction logs on raw disk partitions almost certainly yields better performance.
- Placing extents on raw disk partitions can yield better performance to the degree that doing so reduces swapping. However, if sufficient RAM is available for file system buffers and the shared page cache, better performance may be obtained by placing the extents in the file system.

The use of raw partitions for transaction logs is essential for achieving the highest transaction rates in an update-intensive application because such applications primarily are writing sequentially to the active transaction log. Using raw partitions can as much as double the maximum achievable rate by avoiding the extra file system operations necessary to ensure that each log entry is recorded on the disk.

Because each partition holds a single log or extent, if you place transaction logs in raw partitions, you must provide at least two such partitions so that GemStone can preserve one log when switching to the next. If your application has a high transaction volume, you are likely to find that increasing the number log partitions makes the task of archiving the logs easier.

For information about using raw partitions, see "How To Set Up a Raw Partition" on page 58.

Developing a Failover Strategy

In choosing a failover strategy, consider the following needs:

- Applications that cannot tolerate the loss of committed transactions should mirror the transaction logs (using OS-level tools) and use full transaction logging. A mirrored transaction log on another device allows GemStone to recover from a read failure when it starts up after an unexpected shutdown. The optional full logging mode allows transactions to be rolled forward from a GemStone full backup to recover from the loss of an extent.
- Applications that require rapid recovery from the loss of an extent (that is, without the delay of restoring from a backup) should replicate all extents on other devices, preferably through hardware means, in addition to mirroring transaction logs. Restoring a large repository (many GB) from a backup may take hours.

Hardware replication may provide the best solution if the following points are kept in mind while designing the system:

- Extents benefit from efficiency of both random access (16 KB repository pages) and sequential access. Don't optimize one by compromising the other. Sequential access is important for such operations as garbage collection and making or restoring backups. Use of RAID devices or striped file systems that cannot efficiently support both random and sequential access may reduce overall performance. Simple disk mirroring may be give better results.
- Transaction logs use sequential access exclusively, so the devices can be optimized for that access.
- Avoid volume managers that combine space on multiple physical drives. For GemStone, such configurations may result in *less* efficient access to the repository. The use of raw devices is preferred for transaction logs.

1.2 How To Establish Your Configuration

Configuring the GemStone object server involves the following steps:

1. Gather application specifics about the size of the repository and the number of sessions that will be logged in simultaneously.
2. Plan the operating system resources that will be needed: memory and swap (page) space.

3. Set the size of the GemStone shared page cache and the number of sessions to be supported.
4. Configure the repository extents.
5. Configure the transaction logs.
6. Set GemStone file permissions to allow necessary access while providing adequate security.

Gathering Application Information

When you begin configuring GemStone, be sure to have the following information at hand:

- The number of simultaneous sessions that will be logged in to the repository (in some applications, each user can have more than one session logged in).
- The approximate size of your repository. It's also helpful, but not essential, to know the approximate number of objects in the repository.

This information is central to the sizing decisions that you must make.

Planning Operating System Resources

GemStone needs adequate memory and swap space to run efficiently. It also needs adequate kernel resources—for instance, kernel parameters can limit the size of the shared page cache or the number of sessions that can connect to it.

Estimating Memory Needs

The amount of memory required to run your GemStone server depends mostly on the size of the repository and the number of users who will be logged in to active GemStone sessions at one time. These needs are in addition to the memory required for the operating system and other software.

- The Stone and related processes need between 45 and 325 MB for the configurations shown in Table 1.1 (on page 30). That amount of memory is only for the server processes.
- The shared page cache should be increased in proportion to the overall size of your repository. Typically it should be approximately 10% of the repository size to provide adequate performance, and the larger the better; in most cases, increasing shared page cache provides better performance.

  On a node that is dedicated to running GemStone, we recommend in general that you allocate approximately one-third to one-half of your total system

RAM to the shared page cache. If it is not a dedicated node, you may need to reduce the size to avoid swapping.

- Each Gem session process needs at least 30 MB of memory on the node where it runs. (See the discussion of memory needs for session processes on page 80.) Each Gem process that runs on a remote (client) node also needs about 0.25 MB on the server node for a GemStone page server process that accesses the repository extents.

Estimating Swap Space Needs

To provide reasonable flexibility, the total swap space on your system (sometimes called page space) in general should be at least twice the system RAM. For example, a system with 2 GB of RAM should have at least 4 GB of swap space. The command to find out how much swap space is available (**swap**, **swapinfo**, **pstat**, or **lsvg**) depends on your operating system. Your *GemStone/S 64 Bit Installation Guide* contains an example for your platform.

Swap space should not be on a disk drive that contains any of the GemStone repository extent files. In particular, do not use operating system utilities like **swap** or **swapon** to place part of the swap space on a disk that also contains the GemStone extents or transaction logs.

If you want to determine the additional swap space needed just for GemStone, use the memory requirements derived in the preceding section, including space for the number of sessions you expect. These figures will approximate GemStone's needs beyond the swap requirement for UNIX and other software such as the X Window System.

Estimating File Descriptor Needs

When they start, most GemStone processes attempt to raise their file descriptor limit from the default (soft) limit to the hard limit set by the operating system. In the case of the Stone repository monitor, the processes that raise the limit this way are the Stone itself and two of its child processes, the AIO page server and the Admin GcGem. The Stone uses file descriptors this way:

9 for stdin, stdout, stderr, and internal communication
2 for each user session that logs in
1 for each local extent or transaction log within a file system
2 for each extent or transaction log that is a raw partition
1 for each extent or transaction log that is on a remote node

You can cause the above processes to set a limit less than the system hard limit by setting the GEMSTONE_MAX_FD environment variable to a positive integer. A value of 0 disables attempts to change the default limit.

The shared page cache monitor always attempts to raise its file descriptor limit to equal its maximum number of clients plus five for stdin, stdout, stderr, and internal communication. The maximum number of clients is set by the SHR_PAGE_CACHE_NUM_PROCS configuration option (page 373).

Reviewing Kernel Tunable Parameters

Operating system kernel parameters limit the interprocess communication resources that GemStone can obtain. It's helpful to know what the existing limits are so that you can either stay within them or plan to raise the kernel limits. There are four parameters of primary interest:

- The maximum size of a shared memory segment (typically `shmmax` or a similar name) limits the size of the shared page cache for each repository monitor.

  For information about platform-specific limitations on the size of the shared page cache, refer to Chapter 1 of your *GemStone/S 64 Bit Installation Guide*.

- The maximum number of semaphores per semaphore id limits the number of sessions that can connect to the shared page cache, because each session uses two semaphores. (Typically this parameter is `semmsl` or a similar name, although it is not tunable under all operating systems.)

- The maximum number of users allowed on the system (typically `maxusers` or a similar name) can limit the number of logins and sometimes also is used as a variable in the allocation of other kernel resources by formula. In the latter case, you may need to set it somewhat larger than the actual number of users.

- The hard limit set for the number of file descriptors can limit the total number of logins and repository extents, as described previously.

How you determine the existing limits depends on your operating system. If the information is not readily available, proceed anyway. A later step shows how to verify that the shared memory and semaphore limits are adequate for the GemStone configuration you chose.

Checking the System Clock

The system clock should be set to the correct time. When GemStone opens the repository at startup, it compares the current system time with the recorded checkpoint times as part of a consistency check. A system time earlier than the time

at which the last checkpoint was written may be taken as an indication of corrupted data and prevent GemStone from starting. The time comparisons use GMT. It is not necessary to adjust GemStone for changes to and from daylight savings time.

/opt/gemstone/locks and /opt/gemstone/log

GemStone requires access to two directories under `/opt/gemstone/`:

- `/opt/gemstone/locks` is used for lock files, which among other things provide the names, ports, and other important data used in interprocess communication and reported by **gslist**.

  Under normal circumstances, you should never have to directly access files in this directory. To clear out lock files of processes that exited abnormally, use **gslist -c**.

- `/opt/gemstone/log` is the default location for NetLDI log files, if **startnetldi** does not explicitly specify a location using the **-l** option.

If `/opt/gemstone/` does not exist, GemStone may use `/usr/gemstone/` instead.

Alternatively, you can use the environment variable GEMSTONE_GLOBAL_DIR to specify a different location. Since the files in this location control visibility of GemStone processes to one another, all GemStone processes that interact must use the same directory.

Host Identifier

`/opt/gemstone/locks` (or an alternate directory, as described above) is also the location for a file named `gemstone.hostid`, which contains the unique host identifier for this host. This file is created by the first GemStone process on that host to require a unique identifier, by reading eight bytes from `/dev/random`. This unique hostId is used instead of host name or IP address for GemStone interprocess communication, avoiding issues with multi-homed hosts and changing IP address.

You can access the host identifier for the machine hosting the gem session using the method `System class >> hostId`.

To Set the Page Cache Options and the Number of Sessions

You will configure the shared page cache and the Stone's private page cache according to the size of the repository and the number of sessions that will connect

to it simultaneously. Then use a GemStone utility to verify that the OS kernel will support this configuration.

Shared Page Cache

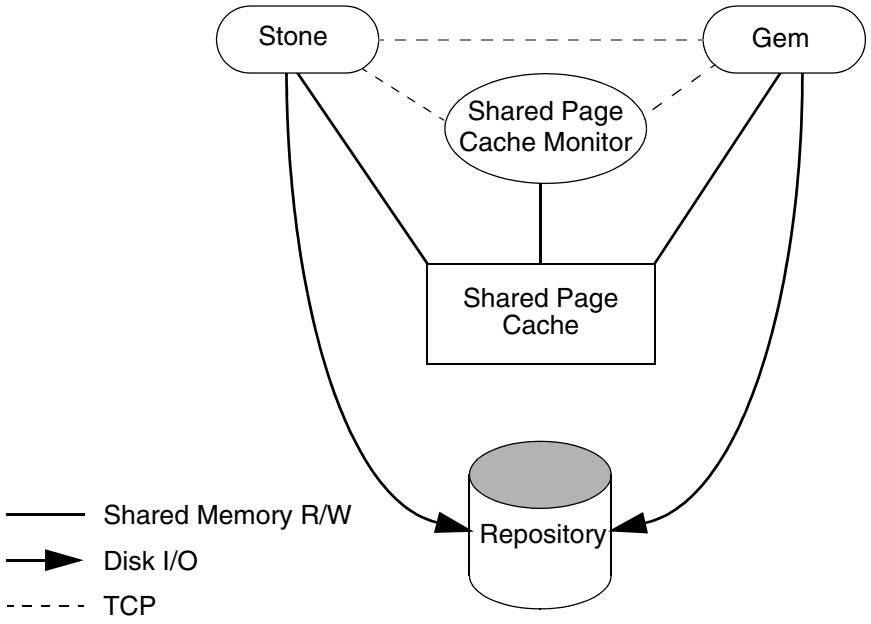
The GemStone shared page cache system consists of two components, the shared page cache itself and a monitor process (`shrpcmonitor`). Figure 1.2 shows the connections between these two and the main GemStone components when GemStone runs on a single node. There is no direct connection between the shared page cache and the repository.

The shared page cache resides in a segment of the operating system's virtual memory that is available to any authorized process. When the Stone repository monitor or a Gem session process needs to access an object in the repository, it first checks to see whether the page containing that object is already in the cache. If the page is already present, the process reads the object directly from shared memory. If the page is not present, the process reads the page from the disk into the cache, where all of its objects also become available to other processes.

The name of the shared page cache monitor ordinarily is derived from the name of the Stone repository monitor and a host-specific ID; for instance, `gs64stone~d7e2174792b1f787`. This host-specific ID is created by the first GemStone process to use a particular host, and remains the same for anything running on that host.

Each shared page cache is associated with exactly one Stone process and repository. A Stone may never have more than one shared page cache on the same node. The Stone spawns the shared page cache automatically during startup, and the shared page cache monitor creates the shared memory region and allocates the semaphores for the system. If other Gem session processes on the same node need to access that repository, they must connect to the same shared page cache and monitor process to ensure cache coherency.

Figure 1.2 Shared Page Cache Configuration



Estimating the Size of the Shared Page Cache

The goal in sizing the shared page cache is to make it large enough to hold the application's working set of objects, thereby reducing disk I/O, while not inducing swapping at the operating system level. Three factors are important in estimating the size:

1. The number of objects in the repository.

   For best performance, the entire object table should be in shared memory. At a minimum, we recommend that you allow room for one-third to one-half of the object table in the cache. Each object uses five bytes in the table.

2. The size of the repository. We recommend that you keep at least 25% of the repository in the cache. Although this factor is application-dependent, increasing the amount in the cache will improve performance.

3. The number of users and the size of their transactions. Again, this factor is specific to your application, depending (among other things) on how frequently users will be committing their transactions.

These rules of thumb provide a basic guideline. Ultimately, the cache size needed depends on the working set size (the number of objects and transactional views) that need to be maintained in the cache to provide adequate performance.

Once your application is running, you can tune the cache size by monitoring the free space. See "Monitoring Performance" on page 173, especially the statistics NumberOfFreeFrames, FramesFromFreeList, and FramesFromFindFree.

Stone's Private Page Cache

As the Stone repository monitor allocates resources to each session, it stores the information in its private page cache. The size of this cache is set by the STN_PRIVATE_PAGE_CACHE_KB configuration option (page 388). The default size of 2 MB is sufficient in almost all circumstances. If you think you might need to adjust this setting, please contact GemStone Technical Support.

Procedure

To configure the shared page cache, follow these steps:

Step 1. Set the SHR_PAGE_CACHE_SIZE_KB configuration option (page 373), using Table 1.1 (on page 30) or your own estimate derived above (remember to convert to KB). For instance, for a 1.5 GB cache:

```
SHR_PAGE_CACHE_SIZE_KB = 1500000;
```

Step 2. If the number of sessions will be greater than 40, increase the STN_MAX_SESSIONS configuration option (page 385) accordingly.

Make sure the SHR_PAGE_CACHE_NUM_PROCS option (page 373) is set to its default (–1), which causes GemStone to calculate a value based on STN_MAX_SESSIONS.

For instance:

```
STN_MAX_SESSIONS = 50;
SHR_PAGE_CACHE_NUM_PROCS = -1;
```

Step 3. Use GemStone's **shmem** utility to verify that your OS kernel supports the chosen cache size and number of processes. The command line is

$GEMSTONE/install/shmem *existingFile cacheSizeKB numProcs*

where

- $GEMSTONE is the directory where the GemStone software is installed.
- *existingFile* is the name of any writable file, which is used to obtain an id (the file is not altered).
- *cacheSizeKB* is the SHR_PAGE_CACHE_SIZE_KB setting.
- *numProcs* is either the SHR_PAGE_CACHE_NUM_PROCS setting or (if that option is set to –1) the greater of 15 or (number of extents + 3).

For instance, for the values used in Steps 1 and 2:

```
% touch /tmp/shmem
% $GEMSTONE/install/shmem /tmp/shmem 1500000 55
% rm /tmp/shmem
```

If **shmem** is successful in obtaining a shared memory segment of sufficient size, no message is printed. Otherwise, diagnostic output will help you identify the kernel parameter that needs tuning. The total shared memory requested includes cache overhead of about 20 bytes per KB of cache space plus about 20 KB per session in SHR_PAGE_CACHE_NUM_PROCS. The actual shared memory segment in this example would be 104865792 bytes (your system might produce slightly different results).

Diagnostics

The shared page cache monitor creates or appends to a log file, *gemStoneNamePid*pcmon.log, in the same directory as the log for the Stone repository monitor. The *Pid* portion of the name is the monitor's process id. In case of difficulty, check for this log.

The operating system kernel must be configured appropriately on each node running a shared page cache. If **startstone** or a remote login fails because the shared cache cannot be attached, check *gemStoneName*.log and *gemStoneNamePid*pcmon.log for detailed information.

The following configuration settings are checked at startup:

- The kernel shared memory resources must be enabled and sufficient to provide the page space specified by SHR_PAGE_CACHE_SIZE_KB plus the cache overhead.

  The kernel semaphore resources must also be sufficient to provide an array of size (SHR_PAGE_CACHE_NUM_PROCS *2) + 1 semaphores.

Use the **shmem** utility to test the settings (see Step 3 on page 41). If multiple Stones are being run concurrently on the same node, each Stone requires a separate set of semaphores and separate semaphore id.

- Sufficient file descriptors must be available at startup to provide one descriptor for each of the SHR_PAGE_CACHE_NUM_PROCS processes plus an overhead of five. Compare your SHR_PAGE_CACHE_NUM_PROCS configuration setting to the operating system file descriptor limit per process. Some operating systems report the descriptor limit in response to the Bash built-in command **ulimit -a**.

  On operating systems that permit it, the shared page cache monitor attempts to raise the descriptor soft limit to the number required. In some cases, raising the limit may require superuser action to raise the hard limit or to reconfigure the kernel.

To Configure the Repository Extents

Configuring the repository extents involves these primary considerations:

- Providing sufficient disk space
- Minimizing I/O contention
- Providing fault tolerance

Estimating Extent Size

When you estimate the size of the repository, allow 10 to 20% for fragmentation. Also allow 0.5 MB of free space for each session that will be logged in simultaneously. In addition, while the application is running, overhead is needed for objects that are created or that have been dereferenced but are not yet removed from the extents. The amount of extent space required for this depends strongly on the particular application and how it is used.

If there is room on the physical disks, and the extents are not at their maximum sizes as specified using DBF_EXTENT_SIZES, then the extents will grow automatically when additional space is needed.

Example 1.1 Extent Size Including Working Space

```
Size of repository            = 1 GB

Free-Space Allowance
  .5 MB * 20 sessions         = 10 MB

Fragmentation Allowance
  1 GB * 15%                  = 150 MB

Total with Working Space      = 1.16 GB
```

If the free space in extents falls below a level set by the STN_FREE_SPACE_THRESHOLD configuration option, the Stone takes a number of steps to avoid shutting down. For information, see "How To Recover from Disk-Full Conditions" on page 240. (The default setting for STN_FREE_SPACE_THRESHOLD is 1 MB; see page 380.)

For planning purposes, you should allow additional disk space for making GemStone backups and for duplicating the repository when upgrading to a new release. A GemStone backup typically occupies 75% to 90% of the total size of the extents, depending on how much space is free in the repository at the time.

Choosing the Extent Location

You should consider the following factors when deciding where to place the extents:

- Keep extents on a spindle different from operating system swap space.
- Where possible, keep the extents and transaction logs on separate spindles.

Specify the location of each extent in the configuration file. The following example uses two raw disk partitions (your partition names will be different):

```
DBF_EXTENT_NAMES = /dev/rdsk/c1t3d0s5, /dev/rdsk/c2t2d0s6;
```

Setting a Maximum Size for an Extent

You can specify a maximum size in MB for each extent through the DBF_EXTENT_SIZES configuration option (page 361). When the extent reaches that size, GemStone stops allocating space in it. If no size is specified, which is the

default, GemStone continues to allocate space for the extent until the file system or raw partition is full.

NOTE

For best performance using raw partitions, the maximum size should be slightly smaller than the size of the partition, so that GemStone can avoid having to handle system errors. For example, for a 2 GB partition, set the size to 1995 MB.

Each size entry is for the corresponding entry in DBF_EXTENT_NAMES (page 360). Use a comma to mark the position of an extent for which you do not want to specify a limit. For example, the following settings are for two extents of 500 MB each in raw partitions.

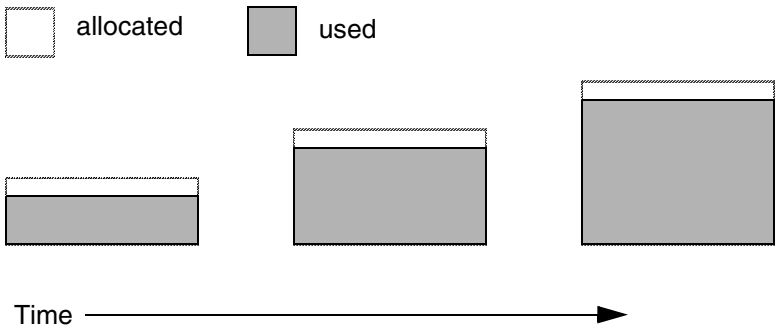
```
DBF_EXTENT_NAMES = /dev/rdsk/c1t3d0s5, /dev/rdsk/c2t2d0s6;
DBF_EXTENT_SIZES = 498, 498;
```

The maximum size of an extent is limited by the operating system and platform, but under no circumstances can be larger than 16 GB. For specific information about system dependencies, see the comment in the configuration file for the parameter `DBF_EXTENT_SIZES`.

Pregrowing Extents to a Fixed Size

Allocating disk space requires a system call that introduces run time overhead. Each time an extent is expanded (Figure 1.3), your application must incur this overhead and then initialize the added extent pages.

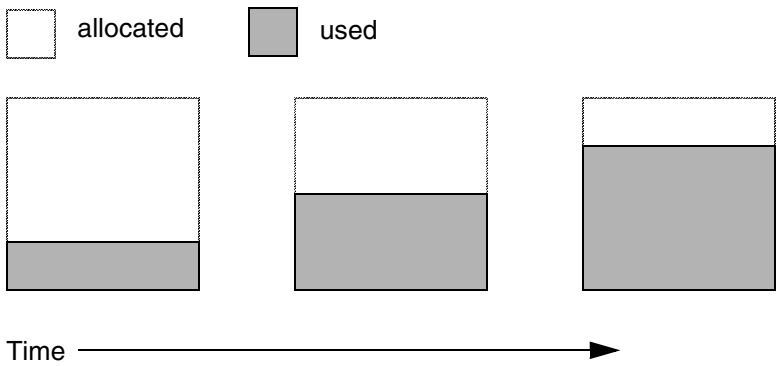
Figure 1.3 Growing an Extent on Demand



You can increase I/O efficiency while reducing file system fragmentation by instructing GemStone to allocate an extent to a predetermined size (called pregrowing it) at startup. When DBF_PRE_GROW is set to True, the Stone repository monitor obtains the necessary space when it creates a new extent or starts with an extent that is smaller than the specified size.

Pregrowing extents avoids repeated system calls to allocate and initialize additional space incrementally. This technique can be used with any number of extents, and with either raw disk partitions or extents in a file system. It is especially useful in performance benchmarking. Pregrowing extents also provides a simple way to reserve space on a disk for a GemStone extent.

Figure 1.4 Pregrowing an Extent



The disadvantages of pregrowing extents are that it takes longer to start GemStone the next time or to add an extent dynamically, and unused disk space allocated to pregrown extents is unavailable for other purposes.

Two configuration options work together to pregrow extents:

- DBF_PRE_GROW (page 361) enables the operation. When DBF_PRE_GROW is set to True, the Stone repository monitor obtains the necessary space when it creates a new extent or starts with an extent that is smaller than the specified size.
- DBF_EXTENT_SIZES (page 361) sets the size limit individually for each extent. For optimal performance, the size should be slightly smaller than the actual size of the disk partition.

To pregrow extents, set both of these configuration options (and remove the comment symbol from DBF_PRE_GROW line). For example:

```
DBF_EXTENT_SIZES = 498, 498;
DBF_PRE_GROW = TRUE;
```

Allocating Data to Multiple Extents

If your application is query-intensive, you should consider dividing the repository into multiple extents and placing each extent on a separate spindle so that accesses can overlap. When GemStone schedules disk writes, it assumes that you have done so. Because several extents can be active at once, putting them on the same spindle limits the maximum update rate and causes updating transactions to have unexpected impact on the response time for queries.

The DBF_ALLOCATION_MODE configuration option (page 360) determines whether GemStone allocates new disk pages to multiple extents by filling each extent sequentially or by balancing the extents using a set of weights you specify. If you have placed each extent on a separate disk drive as recommended, the weighted allocation yields better performance because it distributes disk accesses.

Sequential Allocation

By default, the Stone repository monitor allocates disk resources sequentially by filling one extent to capacity before opening the next extent. (See Figure 1.5 on page 48.) For example, if a logical repository consists of three extents named A, B, and C, then all of the disk resources in A will be allocated before any disk resources from B are used, and so forth. Sequential allocation is used when the DBF_ALLOCATION_MODE configuration option is set to SEQUENTIAL.

Weighted Allocation

For weighted allocation, you use DBF_ALLOCATION_MODE to specify the number of extent pages to be allocated from each extent on each allocation request. The allocations are positive integers in the range 1..40 (inclusive), with each element corresponding to an extent of DBF_EXTENT_NAMES. For example:

```
DBF_EXTENT_NAMES = a.dbf, b.dbf, c.dbf;
DBF_ALLOCATION_MODE = 12, 20, 8;
```

You can think of the total weight of a repository as the sum of the weights of its extents. When the Stone allocates space from the repository, each extent contributes an allocation proportional to its weight.

NOTE

We suggest that you avoid using very small values for weights, such as

"1,1,1". It's more efficient to allocate a group of pages at once, such as "10,10,10", than to allocate single pages repeatedly.

One reason for specifying weighted allocation is to share the I/O load among a repository's extents. For example, you can create three extents with equal weights, as shown in Figure 1.6 (on page 49).

Figure 1.5 Sequential Allocation

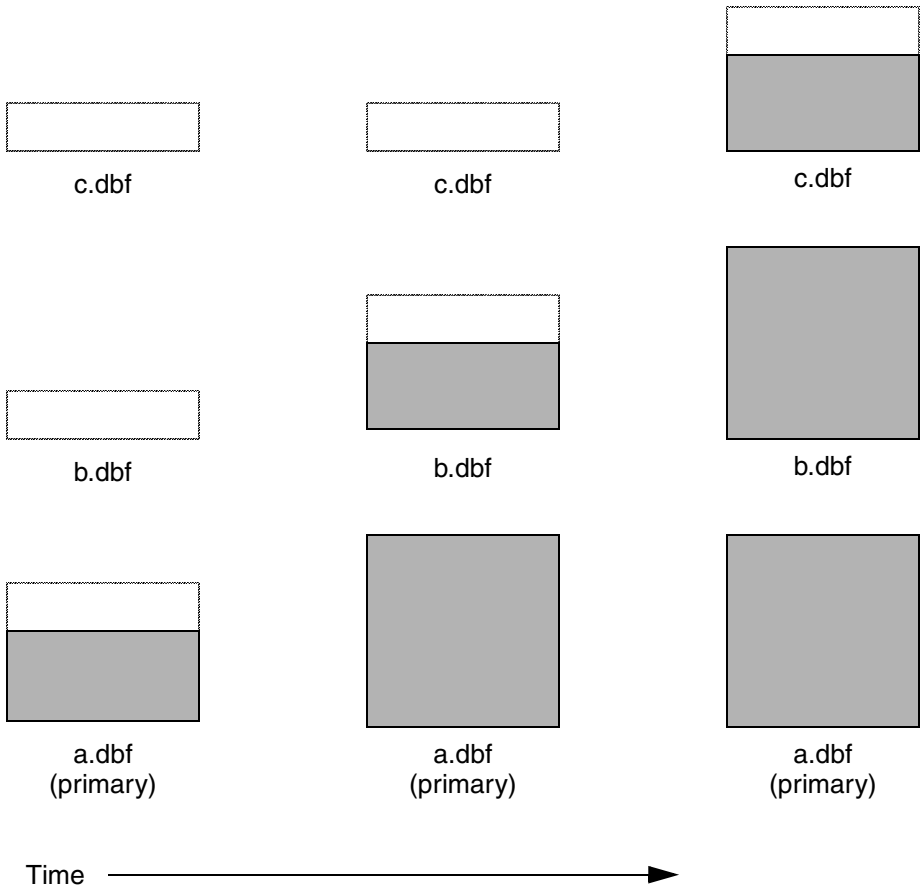
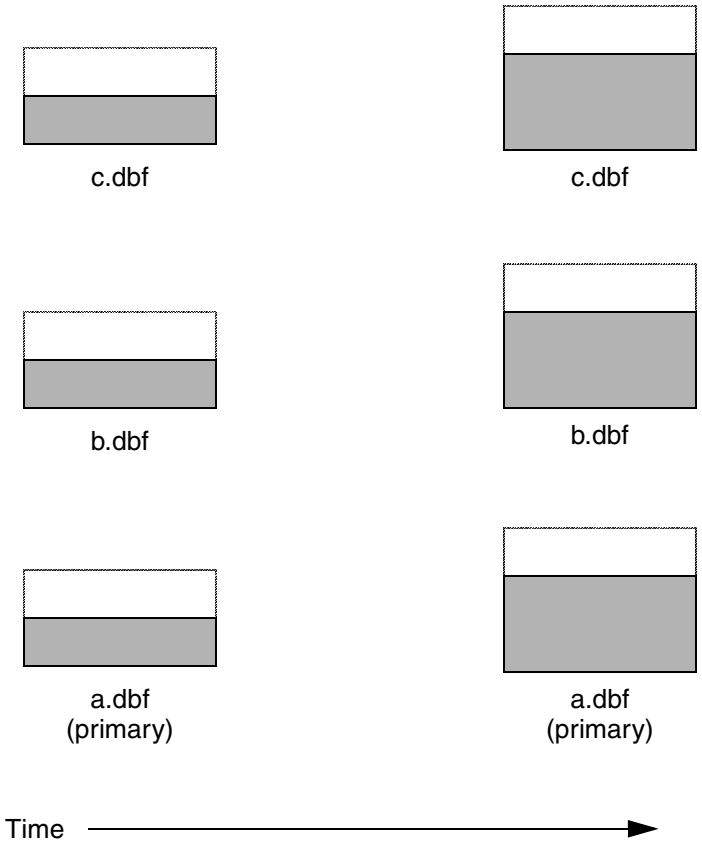


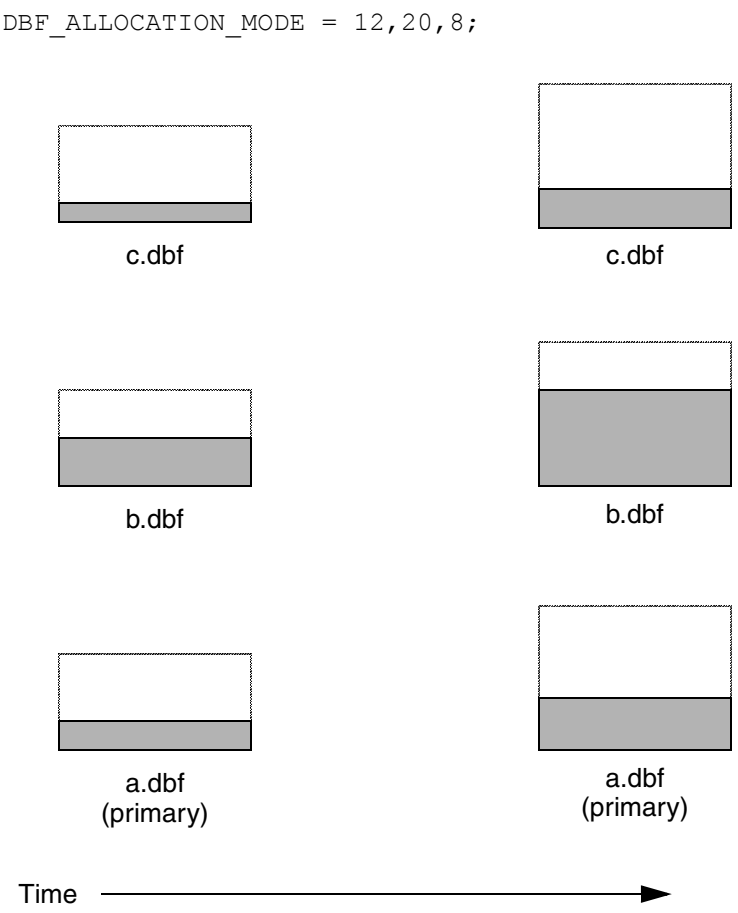
Figure 1.6 Equally Weighted Allocation

```
DBF_ALLOCATION_MODE = 10,10,10;
```



Although equal weights are most common, you can adjust the relative extent weights for other reasons, such as to favor a faster disk drive. For example, suppose we have defined three extents: A, B, and C. If we defined their weights to be 12, 20, and 8 respectively, then for every 40 disk units (pages) allocated, 12 would come from A, 20 from B, and 8 from C. Another way of stating this formula is that because B's weight is 50% of the total repository weight, 50% of all newly-allocated pages are taken from extent B. Figure 1.7 shows the result.

Figure 1.7 Proportionally Weighted Allocation



You can modify the relative extent weights by editing your GemStone configuration file and modifying the values listed for DBF_ALLOCATION_MODE. You can also change DBF_ALLOCATION_MODE to SEQUENTIAL without harming the system. The new values you specify take effect the next time you start the GemStone system.

Effect of Clustering on Allocation Mode

Explicit clustering of objects using instances of ClusterBucket that explicitly specify an extentId takes precedence over DBF_ALLOCATION_MODE. For

information about clustering objects, refer to the *GemStone/S 64 Bit Programming Guide*.

Weighted Allocation for Extents Created at Run Time

Smalltalk methods for creating extents at run time (`Repository>>createExtent:` and `Repository>>createExtent:withMaxSize:`) do not provide a way to specify a weight for the newly-created extent. If your repository uses weighted allocation, the Stone repository monitor assigns the new extent a weight that is the simple average of the repository's existing extents. For instance, if the repository is composed of three extents with weights 6, 10, and 20, the default weight of a newly-created fourth extent would be 12 (36 divided by 3).

To Configure the Transaction Logs

Configuring the transaction logs involves considerations similar to those for extents:

- Choosing a logging mode
- Providing sufficient disk space
- Minimizing I/O contention
- Providing fault tolerance, through the choice of logging mode

Choosing a Logging Mode

GemStone provides two modes of transaction logging:

- *Full logging* provides real-time incremental backup of the repository. Deployed applications should use this mode. All transactions are logged regardless of their size, and the resulting logs can used in restoring the repository from a GemStone backup.
- *Partial logging* is the default mode, and is intended for use during evaluation or early stages of application development. Partial logging is also recommended during bulk loading of the repository. Partial logging allows a simple operation to run unattended for an extended period and permits automatic recovery from system crashes that do not corrupt the repository. Logs created in this mode cannot be used in restoring the repository from a backup.

To enable full transaction logging, change the configuration setting to True and restart the Stone repository monitor:

```
STN_TRAN_FULL_LOGGING = TRUE;
```

CAUTION

The only backups to which you can apply transaction logs are those made while the repository is in full logging mode. If you change to full logging, be sure to make a GemStone backup as soon as circumstances permit.

Changing the logging mode from full to partial logging requires special steps. See "To Change to Partial Logging" on page 254.

For general information about the logging mode and the administrative differences, see "Logging Modes" on page 245.

Estimating the Log Size

How much disk space does your application need for transaction logs? The answer depends on several factors:

- The logging mode that you choose
- Characteristics of your transactions
- How often you archive and remove the logs

If you have configured GemStone for full transaction logging (that is, STN_TRAN_FULL_LOGGING is set to True), you must allow sufficient space to log all transactions until you next archive the logs.

CAUTION

If the Stone exhausts the log space, users will be unable to commit transactions until space is made available.

You can estimate the space required from your transaction rate and the number of bytes modified in a typical transaction. Example 1.2 provides an estimate for an application that expects to generate 4500 transactions a day.

At any point, the method `Repository>>oldestLogFileIdForRecovery` identifies the oldest log file needed for recovery from the most recent checkpoint, if the Stone were to crash. Log files older than the most recent checkpoint (the default maximum interval is 5 minutes) are needed only if it becomes necessary to restore the repository from a backup. Although the older logs can be retrieved from archives, you may want to keep them online until the next GemStone full backup, if you have sufficient disk space.

Example 1.2 Space for Transaction Logs Under Full Logging

```
Average transaction rate                    = 5 per minute

Duration of transaction processing          = 15 hours per day

Average transaction size                    = 5 KB

Archiving interval                          = Daily

Transactions between archives
  5 per minute * 60 minutes * 15 hours      = 4500

Log space (minimum)
  4500 transactions * 5 KB                  = 22 MB
```

If GemStone is configured for partial logging (the default), you need only provide enough space to maintain transaction logs since the last repository checkpoint. Ordinarily, two log files are sufficient: the current log and the immediately previous log. (In partial logging mode, transaction logs are used only after an unexpected shutdown to recover transactions since the last checkpoint.) If you use the default configuration, you should provide space for at least two logs of 2 MB each.

Choosing the Log Location and Size Limit

The considerations in choosing a location for transaction logs are similar to those for extents:

- Keep transaction logs on a different spindle than operating system swap space.
- Where possible, keep the extents and transaction logs on separate spindles—doing so reduces I/O contention while increasing fault tolerance.
- Because update-intensive applications primarily are writing to the transaction log, storing raw data in a disk partition (rather than in a file system) can yield somewhat better performance.

WARNING

Because the transaction logs are needed to recover from a system crash, do NOT place them in directories such as /tmp *that are automatically cleared during power-up.*

GemStone requires at least two log locations (directories or raw partitions) so it can switch to another when the current one is filled. When you set the log locations in the configuration file, you should also check their size limit.

Although the default size of 10 MB is adequate in some situations, update-intensive applications should consider a larger size (at least 25 to 50 MB) to limit the frequency with which logs are switched. Each switch causes a checkpoint to occur, which can impact performance.

NOTE

For best performance using raw partitions, the size setting should be slightly smaller than the size of the partition so GemStone can avoid having to handle system errors. For example, for a 2 GB partition, set it to 1998 MB.

The following example sets up a log in a 2 GB raw partition and a directory of 50 MB logs in the file system. This setup is a workable compromise when the number of raw partitions is limited. The file system logs give the administrator time to archive the primary log when it is full.

```
STN_TRAN_LOG_DIRECTORIES = /dev/rdsk/c4d0s2,
/user3/tranlogs;
STN_TRAN_LOG_SIZES = 1998, 50;
```

All of the transaction logs must reside on Stone's node.

To Configure Server Response to Gem Fatal Errors

The Stone repository monitor is configurable in its response to a fatal error detected by a Gem session process. By default, the Stone halts and dumps a core image if it receives notification from a Gem that the Gem process died with a fatal error that would cause the Gem to dump core. By stopping both the Gem and the Stone at this point, the possibility of repository corruption is minimized. During application development, it may be helpful to know exactly what the Stone was doing when the Gem went down.

For deployed production systems, we recommend that you change the default in the Stone's configuration file so that the Stone will attempt to keep running:

```
STN_HALT_ON_FATAL_ERR = FALSE;
```

To Set File Permissions for the Server

The primary consideration in setting file permissions for the Server is to protect the repository extents. All reads and writes should be done through GemStone

repository executables: the Stone and its child processes (the shared page cache monitor, AIO page server, Admin and Reclaim GcGems, Symbol Gem, Page Manager, and Free Frame Page Server) and the Gem session processes. Chapter 3 describes the use of additional page servers to read and write extents in networked systems.

Recommended: Use the Setuid Bit

The tightest repository security is obtained by having the extents and the repository executables owned by a single UNIX account, using the UNIX setuid bit (S bit) on the executable files, and making the extents writable only by that account. The S bit causes a process to belong to the owner you specify for the file.

Table 1.2 shows the recommended file settings, where *gsadmin* and *gsgroup* can be any ordinary UNIX account and group (do NOT use the root account for this purpose). The person who starts the repository monitor (Stone) must be logged in as *gsadmin* or have execute permission. The Stone and shared page cache will belong to the owner you specify for the files.

Table 1.2 Recommended Resource and Process Permissions for the Server

Resource or Process[a]	Protection Mode	File Owner	File Group	Process Runs As	Comments
repository extents	`-rw-------`	gsadmin	gsgroup		Users read and write repository through GemStone processes. The Stone sets up the page cache in shared memory.
shared memory	`-rw-rw----`	gsadmin	gsgroup		
stoned	`-r-sr-xr-x`	gsadmin	gsgroup	gsadmin	AIO page server and GC Gem are spawned by stoned and can access repository as the *gsadmin* account.
pgsvrmain	`-r-sr-xr-x`	gsadmin	gsgroup	gsadmin	
gem	`-r-sr-xr-x`	gsadmin	gsgroup	gsadmin	

[a] Ownership and permissions for the netldid executable depend on the authentication mode chosen and are discussed in Chapter 3.

If you are logged in as root when you run the GemStone installation program, it offers to set file protections in the manner described in Table 1.2. To set them manually, do the following as root:

```
# cd $GEMSTONE/sys
# chmod u+s gem pgsvr pgsvrmain stoned
# chown gsadmin gem pgsvr pgsvrmain stoned
# cd $GEMSTONE/data
# chmod 600 extent0.dbf
# chown gsadmin extent0.dbf
```

The protection mode for the shared memory segment is fixed in GemStone.

You must take similar steps to provide access for repository clients, which are presented in Chapter 2. See "To Set Ownership and Permissions for Session Processes" on page 82.

Alternative: Use Group Write Permission

For sites that prefer not to use the setuid bit, the alternative is to make the extents writable by a particular UNIX group and have all users belong to that group. That group must be the primary group of the person who starts the Stone (that is, the one listed in `/etc/passwd`). Do the following, where *gsgroup* is a group of your choice:

```
% cd $GEMSTONE/data
% chmod 660 extent0.dbf
% chgrp gsgroup extent0.dbf
```

Sites that run the linked version of GemBuilder may also prefer to use this protection so that fileouts and other I/O operations that do not read or write the repository will be done using the individual user's id instead of the single *gsadmin* account.

Access to Other Server Files

GemStone creates log files and other special files in several locations, which are described below. In a multi-user environment, the protection of these resources must be such that the appropriate file can be created or updated in response to actions by several users.

`/opt/gemstone`	All users should be able to read files in the directory `/opt/gemstone/locks` on each node (or an equivalent location, as discussed on page 38). Users who will start a Stone or NetLDI process require read, write and execute access to `/opt/gemstone/log` and `/opt/gemstone/locks` on each node.
`system.conf`	The user who owns the Stone process must have write permission for the Stone configuration file, which by default is `$GEMSTONE/data/system.conf`. If certain configuration changes are made while the Stone is running, the Stone updates that file. For instance, the Stone must record run-time changes such as those made by `Repository>>createExtent:` so that it can restart later in the correct configuration.

1.3 How To Set Up a Raw Partition

WARNING

Using raw partitions requires extreme care. Overwriting the wrong partition destroys existing information, which in certain cases can make data on the entire disk inaccessible.

The instructions in this section are incomplete intentionally. You will need to work with your system administrator to locate a partition of suitable size for your extent or transaction log. Consult the system documentation for guidance as necessary.

You can mix file system-based files and raw partitions in the same repository, and you can add a raw partition to existing extents or transaction log locations. The partition reference in `/dev` must be readable and writable by anyone using the repository, so you should give the entry in `/dev` the same protection as you would use for the corresponding type of file in the file system.

The first step is to find a partition (raw device) that is available for use. Depending on your operating system, a raw partition may have a name like `/dev/rdsk/c1t3d0s5`, `/dev/rsd2e`, or `/dev/vg03/rlvol1`. Most operating systems have a utility or administrative interface that can assist you in identifying existing partitions; some examples are **prtvtoc**, **dkinfo**, and **vgdisplay**. A partition is available if all of the following are true:

- It does not contain the root (/) file system (on some systems, the root volume group).
- It is not on a device that contains swap space.
- Either it does not contain a file system or that file system can be left unmounted and its contents can be overwritten.
- It is not already being used for raw data.

When you select a partition, make sure that any file system tables, such as `/etc/vfstab`, do not call for it to be mounted at system boot. If necessary, unmount a file system that is currently mounted and edit the system table. Use **chmod** and **chown** to set read-write permissions and ownership of the special device file the same way you would protect a repository file in a file system. For example, set the permissions to 600, and set the owner to the GemStone administrator.

If the partition will contain the primary extent (the first or only one listed in DBF_EXTENT_NAMES), initialize it by using the GemStone **copydbf** utility to copy an existing repository extent to the device. The extent must not be in use when you

copy it. If the partition already contains a GemStone file, first use **removedbf** to mark the partition as being empty.

Partitions for transaction logs do not need to be initialized, nor do secondary extents into which the repository will expand later.

Sample Raw Partition Setup

The following example configures GemStone to use the raw partition `/dev/rsd2d` as the repository extent.

Step 1. If the raw partition already contains a GemStone file, mark it as being empty. (The **copydbf** utility will not overwrite an existing repository file.)

```
% removedbf /dev/rsd2d
```

Step 2. Use **copydbf** to install a fresh extent on the raw partition. (If you copy an existing repository, first stop any Stone that is running on it.)

```
% copydbf $GEMSTONE/bin/extent0.dbf /dev/rsd2d
```

Step 3. As root, change the ownership and the permission of the partition special device file in `/dev` to what you ordinarily use for extents in a file system. For instance:

```
# chown gsAdmin /dev/rsd2d
# chmod 600 /dev/rsd2d
```

You should also consider restricting the execute permission for `$GEMSTONE/bin/removedbf` and `$GEMSTONE/bin/removeextent` to further protect your repository. In particular, these executable files should not have the setuid (S) bit set.

Step 4. Edit the Stone's configuration file to show where the extent is located:

```
DBF_EXTENT_NAMES = /dev/rsd2d;
```

Step 5. Use **startstone** to start the Stone repository monitor in the usual manner.

Changing Between Files and Raw Partitions

This section tells you how to change your configuration by moving existing repository extent files to raw partitions or by moving existing extents in raw partitions to files in a file system. You can make similar changes for transaction logs.

Moving an Extent to a Raw Partition

To move an extent from the file system to a raw partition, do this:

Step 1. Define the raw disk partition device. Its size should be at least 1 to 2 MB larger than the existing extent file.

Step 2. Stop the Stone repository monitor.

Step 3. Edit the repository's configuration file, substituting the device name of the partition for the file name in DBF_EXTENT_NAMES (page 360).

Set DBF_EXTENT_SIZES for this extent to be 1 to 2 MB smaller than the size of the partition.

Step 4. Use **copydbf** to copy the extent file to the raw partition. (If the partition previously contained a GemStone file, first use **removedbf** to mark it as unused.)

Step 5. Restart the Stone.

Moving an Extent to the File System

The procedure to move an extent from a raw partition to the file system is similar:

Step 1. Stop the Stone repository monitor.

Step 2. Edit the repository's configuration file, substituting the file pathname for the name of the partition in DBF_EXTENT_NAMES.

Step 3. Use **copydbf** to copy the extent to a file in a file system, then set the file permissions to the ones you ordinarily use.

Step 4. Restart the Stone.

Moving Transaction Logging to a Raw Partition

To switch from transaction logging in the file system to logging in a raw partition, do this:

Step 1. Define the raw disk partition. If you plan to copy the current transaction log to the partition, its size should be at least 1 to 2 MB larger than current log file.

Step 2. Stop the Stone repository monitor.

Step 3. Edit the repository's configuration file, substituting the device name of the partition for the directory name in STN_TRAN_LOG_DIRECTORIES (page 391). Make sure that STN_TRAN_LOG_SIZES for this location is 1 to 2 MB smaller than the size of the partition.

Step 4. Use **copydbf** to copy the current transaction log file to the raw partition. (If the partition previously contained a GemStone file, first use **removedbf** to mark it as unused.)

You can determine the current log from the last message "Creating a new transaction log" in the Stone's log. If you don't copy the current transaction log, the Stone will open a new one with the next sequential fileId, but it may be opened in another location specified by STN_TRAN_LOG_DIRECTORIES.

Step 5. Restart the Stone.

Moving Transaction Logging to the File System

The procedure to move transaction logging from a raw partition to the file system is similar:

Step 1. Stop the Stone repository monitor.

Step 2. Edit the repository's configuration file, substituting a directory pathname for the name of the partition in STN_TRAN_LOG_DIRECTORIES.

Step 3. Use **copydbf** to copy the current transaction log to a file in the specified directory. The **copydbf** utility will generate a file name like tranlog*nnn*.dbf, where *nnn* is the internal fileId of that log.

Step 4. Restart the Stone.

1.4 How To Access the Server Configuration at Run Time

GemStone provides several methods in class System that let you examine, and in certain cases modify, the configuration parameters at run time from Smalltalk.

To Access Current Settings at Run Time

Class methods in category Configuration File Access let you examine the system-Stone configuration. The following access methods all provide similar server information:

`stoneConfigurationReport`
: Returns a SymbolDictionary whose keys are the names of configuration file parameters, and whose values are the current settings of those parameters in the repository monitor process.

`configurationAt:` *aName*
: Returns the value of the specified configuration parameter, giving preference to the current session process if the parameter applies to a Gem.

`stoneConfigurationAt:` *aName*
: Returns the value of the specified configuration parameter from the Stone process, or returns `nil` if that parameter is not applicable to a Stone.

(The corresponding methods for accessing a session configuration are described on page 85.)

Here is a partial example of the Stone configuration report:

```
topaz 1> printit
System stoneConfigurationReport asReportString
%
#SHR_SPIN_LOCK_COUNT    5000
#StnDisableLoginFailureTimeLimit        15
#StnDisableLoginFailureLimit    15
#SHR_PAGE_CACHE_LOCKED  false
...
```

Keys in mixed capitals and lowercase, such as `SpinLockCount`, are internal run-time parameters.

To Change Settings at Run Time

The class method `System class>>configurationAt:` *aName* `put:` *aValue* in category Runtime Configuration Access lets you change the value of the internal run-time parameters in Table 1.3 if you have the appropriate privileges. The parameters that can be changed are those for which `ConfigurationParameterDict at:` *aName* returns a negative SmallInteger. All changeable parameters require that *aValue* be a SmallInteger.

CAUTION

Avoid changing configuration parameters unless there is a clear reason for doing so. Incorrect settings can have serious adverse effects on GemStone performance. For additional guidance about run-time changes to specific parameters, see Appendix A, "GemStone Configuration Options."

Table 1.3 Server Configuration Parameters Changeable at Run Time

Configuration File Option	Internal Parameter
SHR_SPIN_LOCK_COUNT	#SpinLockCount [a]
STN_ADMIN_GC_SESSION_ENABLED	#StnAdminGcSessionEnabled [b]
STN_CHECKPOINT_INTERVAL	#StnCheckpointInterval [a]
STN_COMMIT_QUEUE_THRESHOLD	#StnCommitQueueThreshold [a]
STN_CR_BACKLOG_THRESHOLD	#StnCrBacklogThreshold [a]
STN_DISABLE_LOGIN_FAILURE_LIMIT	#StnDisableLoginFailureLimit [c]
STN_DISABLE_LOGIN_FAILURE_TIME_LIMIT	#StnDisableLoginFailureTimeLimit [c]
STN_DISKFULL_TERMINATION_INTERVAL	#StnDiskFullTerminationInterval [a]
STN_EPOCH_GC_ENABLED	#StnEpochGcEnabled [b]
STN_FREE_SPACE_THRESHOLD	#StnFreeSpaceThreshold
STN_GEM_ABORT_TIMEOUT	#StnGemAbortTimeout [a]
STN_GEM_LOSTOT_TIMEOUT	#StnGemLostOtTimeout [a]
STN_GEM_TIMEOUT	#StnGemTimeout [a]
STN_HALT_ON_FATAL_ERR	#StnHaltOnFatalErr [a]
STN_LOG_LOGIN_FAILURE_LIMIT	#StnLogLoginFailureLimit [c]
STN_LOG_LOGIN_FAILURE_TIME_LIMIT	#StnLogLoginFailureTimeLimit [c]

Table 1.3 Server Configuration Parameters Changeable at Run Time (Continued)

Configuration File Option	Internal Parameter
STN_LOOP_NO_WORK_THRESHOLD	#StnLoopNoWorkThreshold [a]
STN_MAX_AIO_RATE	#StnMntMaxAioRate [a]
STN_MAX_VOTING_SESSIONS	#StnMaxVotingSessions [a]
STN_NUM_GC_RECLAIM_SESSIONS	#StnNumGcReclaimSessions [b]
STN_OBJ_LOCK_TIMEOUT	#StnObjLockTimeout
STN_PAGE_MGR_COMPRESSION_ENABLED	#StnPageMgrCompressionEnabled [a]
STN_PAGE_MGR_PRINT_TIMEOUT_THRESHOLD	#StnPageMgrPrintTimeoutThreshold [a]
STN_PAGE_MGR_REMOVE_MAX_PAGES	#StnPageMgrRemoveMaxPages [a]
STN_PAGE_MGR_REMOVE_MIN_PAGES	#StnPageMgrRemoveMinPages [a]
STN_REMOTE_CACHE_PGSVR_TIMEOUT	#StnRemoteCachePgsvrTimeout [d]
STN_REMOTE_CACHE_TIMEOUT	#StnRemoteCacheTimeout [a]
STN_SHR_TARGET_PERCENT_DIRTY	#ShrPcTargetPercentDirty [a]
STN_TRAN_LOG_DEBUG_LEVEL	#StnTranLogDebugLevel [d]
STN_SIGNAL_ABORT_CR_BACKLOG	#StnSignalAbortCrBacklog [b]
STN_TRAN_LOG_LIMIT	#StnTranLogLimit [a]
STN_TRAN_Q_TO_RUN_Q_THRESHOLD	#StnTranQToRunQThreshold [a]
(none)	#StnLoginsSuspended [d]

[a] Can be changed only by SystemUser.

[b] Requires GarbageCollection privilege.

[c] Requires OtherPassword privilege.

[d] Requires SystemControl privilege.

The following example first obtains the value of #StnAdminGcSessionEnabled. This value can be changed at run time by a user with GarbageCollection privilege:

```
topaz 1> printit
System configurationAt: #StnAdminGcSessionEnabled
%
1
topaz 1> printit
System configurationAt: #StnAdminGcSessionEnabled put: 0
%
0
```

For more information about these methods, see the comments in the image.

1.5 How To Tune Server Performance

There are a number of configuration options by which you can tune the GemStone server. These options can help make better use of the shared page cache, reduce swapping, and control disk activity caused by repository checkpoints.

To Tune the Shared Page Cache

Two configuration options can help you tailor the shared page cache to the needs of your application: SHR_PAGE_CACHE_SIZE_KB and SHR_SPIN_LOCK_COUNT.

You may also want to consider object clustering within Smalltalk as a means of increasing cache efficiency.

Adjusting the Cache Size

Adjust the SHR_PAGE_CACHE_SIZE_KB configuration option (page 373) according to the total number of objects in the repository and the number accessed at one time. For best performance, the entire object table should be in shared memory. At a minimum, we recommend that the shared page cache should be large enough to hold one-third to one-half of the object table and all the pages on which currently used objects reside.

In general, the more of your repository you can hold in your cache, the better your performance will be. The size of the cache should not exceed one-half of your physical memory.

You should review the configuration recommendations given earlier (“Estimating the Size of the Shared Page Cache” on page 40) in light of your application’s design

and usage patterns. Estimates of the number of objects queried or updated are particularly useful in tuning the cache.

You can use the shared page cache statistics for a running application to monitor the load on the cache. In particular, the statistics FreeFrameCount and FramesFromFindFree may be useful, as well as FramesFromFreeList.

Matching Spin Lock Limit to Number of Processors

The SHR_SPIN_LOCK_COUNT configuration option (page 373) specifies the number of times a process should attempt to obtain a lock in the shared page cache using the spin lock mechanism before resorting to setting a semaphore and sleeping. We recommend you leave SHR_SPIN_LOCK_COUNT set to –1 (the default), which causes GemStone to determine whether multiple processors are installed and set the parameter accordingly.

Clustering Objects That Are Accessed Together

Appropriate clustering of objects by the application can allow a smaller shared page cache by reducing the number of data pages in use at once. For general information about clustering objects, see the *GemStone/S 64 Bit Programming Guide*.

To Reduce Excessive Swapping

Be careful not to make the shared page cache so large that it forces excessive swapping. If your node is dedicated to running GemStone, our general recommendation (as given on page 40) is that you use up to one-half of its RAM for the cache. If it is not a dedicated node, you may need to limit the cache size to something smaller than one-half of the RAM. When the node's memory is also used for non-GemStone processes, your shared page cache may need to be swapped, which would affect performance adversely.

Excessive swapping also can be caused by the need to awaken (and swap in) sleeping sessions that are outside of a transaction. When the Stone repository monitor reaches its configured limit, it sends a SignaledAbort message to the sleeping session, causing it to be swapped back into memory. Each such session must awaken long enough to update its view of the repository.

You can reduce this type of swapping activity by increasing the STN_SIGNAL_ABORT_CR_BACKLOG configuration option (page 390). For example, you might determine a desired interval between SignaledAbort messages, and then use your application's commit rate to calculate the setting of STN_SIGNAL_ABORT_CR_BACKLOG. You may need to take the following additional steps:

- Increase the STN_PRIVATE_PAGE_CACHE_KB configuration option to (STN_SIGNAL_ABORT_CR_BACKLOG + 10) / 30 MB.
- Increase the size of the shared page cache. The default backlog of 20 commit records requires about 1 MB, assuming that a typical small transaction occupies about 50 KB.

If your configuration uses multiple extents in the file system, you may be able to reduce swapping by limiting the size of file system buffers. Some operating systems do not support this restriction.

To Control Checkpoint Frequency

Each checkpoint guarantees that the committed state of the repository has been written to the extent files. If the checkpoints interfere with other GemStone activity, you may want to adjust their frequency.

- In full transaction logging mode, most checkpoints are determined by the STN_CHECKPOINT_INTERVAL configuration option, which by default is five minutes (see page 376). A few Smalltalk methods, such as `Repository>>fullBackupTo:`, force a checkpoint at the time they are invoked. A checkpoint also is performed each time the Stone begins a new transaction log, so you may want to increase the size of these logs to reduce the frequency of checkpoints.
- In partial logging mode, checkpoints also are triggered by any transaction that is larger than STN_TRAN_LOG_LIMIT, which sets the size of the largest entry that is to be appended to the transaction log (see page 391). The default limit is 1 MB of log space. If checkpoints are too frequent in partial logging mode, it may help to raise this limit. Conversely, during bulk loading of data with large transactions, it may be desirable to lower this limit to avoid creating large log files.

  For information about tuning STN_TRAN_LOG_LIMIT in partial logging mode, see the discussion of the CheckpointCount cache statistic on page 478.

A checkpoint also occurs each time the Stone repository monitor is shut down gracefully, as by invoking **stopstone** or `System class>>shutDown`. This checkpoint permits the Stone to restart without having to recover from transaction logs. It also permits extent files to be copied in a consistent state.

While less frequent checkpoints may improve performance, they may extend the time required to recover after an unexpected shutdown.

Suspending Checkpoints

You can call the method `System class>>suspendCheckpointsForMinutes:` to suspend checkpoints for a given number of minutes, or until `System class>>resumeCheckpoints` is executed. (To execute these Smalltalk methods, you must have the required GemStone privilege, as described in Chapter 6, "User Accounts and Security".)

Under normal circumstances, this approach is used only to allow online extent backups to complete. For details on how to suspend and resume checkpoints, see "How To Make an Online Extent Backup" on page 259.

Page Servers

GemStone uses page servers for three purposes:

- To write dirty pages to disk.
- To transfer pages from the Stone host to the shared page cache host, if different.
- To add free frames to the free frame list, from which a Gem can take as needed.

Page servers referred to as *AIO page servers* perform all three functions. By default, at least one AIO page server is running at all times, although larger systems are likely to need additional page servers.

In addition, by default, one *free frame page server* is running. Free frame page servers are dedicated only to the third task listed above: adding free frames to the free list. In some cases, increasing the number of free frame page servers can improve overall system performance. For example, if Gems are performing many operations requiring writing pages to disk, the AIO page server may have to spend all its time writing pages, never getting a chance to add free frames to the free list. Alternatively, if Gems are performing operations that require only reading, the AIO page server will see no dirty frames in the cache—the signal that prompts it to take action. In that case, it may sleep for a second, even though free frames are present in the cache and need to be added to the free list.

To Add AIO Page Servers

By default the Stone spawns a single page server process on its local node to perform asynchronous I/O (AIO) between the shared page cache and the extents. This page server ordinarily is the process that updates extents on the local node during a checkpoint. (In some cases, the Stone may use additional page servers temporarily during startup to pregrow multiple extents.)

If your configuration has multiple extents on separate disk spindles, consider increasing the number of AIO page servers in use during ordinary operation. You can do this by changing the STN_NUM_LOCAL_AIO_SERVERS configuration option (page 386).

For multiple page servers to improve performance, they must be able to execute at the same time and write to disk at the same time. If you have only one CPU, or your extents are on a single disk spindle, multiple AIO page servers will not be able to write pages out faster than a single page server.

Do You Need Free Frame Page Servers?

A Gem can get free frames either from the free list (the quick way), or, if sufficient free frames are not listed, by scanning the shared page cache for a free frame instead. (What constitutes sufficient free frames is determined by the configuration parameter GEM_FREE_FRAME_LIMIT; for details, see page 363.

If a Gem has to spend a large proportion of its time scanning the shared page cache, its performance may be unacceptable. Under these circumstances, extra free frame page servers can sometimes help. On a single-CPU system, one extra free frame page server might be all that's required; for systems with multiple CPUs, you may wish to start one at a time, checking statistics, until the problem is resolved.

By default, when you start the Stone, it spawns one free frame page server process. Certain cache statistics can help you determine whether additional free frame page servers will improve system performance. (For details about these and other statistics, see "Cache Statistics" on page 475.)

- If Gems have to scan the shared page cache for free frames, the cache statistic FramesFromFindFree will be greater than zero. If this is the case—especially if it significantly greater—consider starting one or more free frame page servers.
- If the FreeFrameCount is consistently lower than the FreeFrameLimit, a free frame page server might help (though other factors also enter into play).

If FramesAddedToFreeList rises significantly after starting a free frame page server, or if FramesFromFindFree is reduced to zero (or near zero), the new page server has indeed benefited you.

To Add Free Frame Page Servers

You can change the number of free frame page server processes that will be started when the shared page cache is created by setting a configuration parameter, SHR_NUM_FREE_FRAME_SERVERS.

Default: 1
Minimum: 1
Maximum: 30

Process Free Frame Caches

There is a communication overhead involved in getting free frames from the free frame list for scanning. To optimize this, you can configure the Gems and their remote page servers to add or remove multiple free frames from a free frame cache to the free frame list in a single operation.

When using the free frame cache, the Gem or remote page server process removes enough frames from the free list to refill the cache in a single operation. When adding frames to the free list, the process does not add them until the cache is full.

You can control the size of the Gem and remote page server free frame caches by setting the configuration parameters GEM_FREE_FRAME_CACHE_SIZE and GEM_PGSVR_FREE_FRAME_CACHE_SIZE, respectively.

For each of these, a value of 0 disables the free frame cache (the Gem or remote page server process acquires frames one at a time). A value of -1 means use the default value: 0 (for shared page caches less than 100 MB) or 10 (for shared page caches of 100 MB or greater).

Default: -1
Minimum: -1
Maximum: 63

1.6 How To Run a Second Repository

You can run more than one repository on a single node—for example, separate production and development repositories. There are several points to keep in mind:

- Each repository requires its own Stone repository monitor process, extent files, transaction logs, and configuration file. (Each Stone will also start its own shared page cache monitor and a set of other processes, as described on page 27.)

- You must give each Stone a unique name at startup. That name is used to identify the server and to maintain separate log files. Users will connect to the repository by specifying the Stone's name.
- A single NetLDI serves all Stones and Gem session processes on a given node, provided both Stones are running the same version.
- Multiple Stones can share a single installation directory, provided that you create separate repository extents, transaction logs, and configuration files. If performance is a concern, the first step should be to isolate each Stone's data directory and the system swap space on separate drives. Then, review the discussion "Recommendations About Disk Usage" on page 32.

The following example shows the steps necessary to create a separate repository for application development (identified here by the prefix `dev`). This repository will run in parallel with the initial repository that you installed by following the instructions in the *GemStone/S 64 Bit Installation Guide.*

In this example, we use the `$GEMSTONE` installation tree to avoid having to duplicate files that can be shared. To reduce I/O contention, we create a separate data directory on another disk.

Step 1. Copy a fresh repository extent and configuration file to the new data directory. Make the files writable by the development group.

```
% mkdir /user2/devdata
% cd /user2/devdata
% cp $GEMSTONE/bin/extent0.dbf .
% cp $GEMSTONE/bin/initial.config system.conf
% chmod ug+w extent0.dbf system.conf
```

Step 2. Edit the new configuration file so that it specifies the proper extent file. Change the transaction log directories to the new data directory.

```
DBF_EXTENT_NAMES = /user2/devdata/extent0.dbf;
STN_TRAN_LOG_DIRECTORIES = /user2/devdata,
/user2/devdata;
```

Step 3. Set the environment variable GEMSTONE_SYS_CONF so it points to the new configuration file. GemStone will use the new configuration file in place of the default, which is `$GEMSTONE/data/system.conf`. For example:

```
$ GEMSTONE_SYS_CONF=/user2/devdata/system.conf
$ export GEMSTONE_SYS_CONF
```

Step 4. Start the Stone for the development repository, giving it the name *devserver30*. GemStone will create log files with that server name as the prefix.

```
$ startstone devserver30
```

Step 5. Start linked Topaz, then set the GemStone name to *devserver30* and log in as DataCurator:

```
% topaz -l
topaz> set gemstone devserver30
topaz> set username DataCurator
topaz> login
GemStone Password?
successful login
topaz 1>
```

At this point, you are logged in and you can begin installing user accounts for developers. However, the repository is the one in `devdata`. Any changes that you commit to this repository will not affect `$GEMSTONE/data/extent0.dbf`, and existing applications can use the latter repository independently.

1.7 How To Operate a Duplicate Server/Warm Standby

Some customers may want to keep a duplicate of a production GemStone server running almost in parallel as a "warm" standby. The duplicate continually runs in restore mode, restoring each transaction log from the production server after the log is closed. If anything goes wrong with the primary production server, the warm standby can be brought into use very quickly.

This section tells how to set up the duplicate server and restore the logs. For general information about restoring backups and transaction logs, see "How To Restore from a Smalltalk Full Backup" on page 269. This discussion assumes you are familiar with that procedure.

An important point to remember is that the transaction logs copied from the production server, called the *archive logs* here, must be kept separate from the transaction logs created by the duplicate server. You can do that by using different log directories or different file name prefixes.

To operate a warm standby, the primary system must be running in full logging mode, as described in "Choosing a Logging Mode" on page 51.

Setup and run the warm standby

Step 1. Install the duplicate server. For fault tolerance, it's best to do a complete GemStone installation on a second node.

Step 2. Decide on a naming convention or location that you will use on the duplicate server to keep the archive logs separate from those being created by the duplicate server itself. For instance, if both Stones use the default prefix of *tranlog*, you might copy `tranlog123.dbf` on the production server to `$GEMSTONE/data/prodtranlog123.dbf` on the duplicate server.

Step 3. Make an extent copy backup, or a programmatic full backup of the primary system. (Instructions begin on page 257.) You'll have to do this at least once, when you start this project; however, regular backups will simplify matters when you need to synchronize the primary and the standby systems.

Step 4. Copy the extent backups, or restore the full backup, into the duplicate server. If you use extent copies, use the -R option to start the Stone, which causes the Stone to enter restore mode. For instance, **startstone** –**R**.

Step 5. As each transaction log completes on the primary system, move the log to a file system accessible to the warm standby GemStone installation. Avoid moving these logs into the transaction log directory that the warm standby uses for its own transaction logs.

Wait several seconds after the new log is created before copying the old log, to ensure the completion of any asynchronous writes.

You can limit each transaction log to a tolerable amount of information either by limiting transaction log size or by starting a new log at regular time intervals:

- ***Size-based:*** Limit the transaction log size, as described in "Choosing the Log Location and Size Limit" on page 53. When a transaction log grows to the specified limit, GemStone starts a new transaction log.
- ***Time-based:*** On your primary system, run a script at regular intervals that terminates the current transaction log and starts a new one, using the method `System class >> startNewLog`.

Step 6. On the warm standby, restore the transaction logs as they are available.

Since the transaction logs for the primary are not in the STN_TRAN_LOG_DIRECTORIES of the standby, you will set the archive log

directory for these tranlogs. You can set the archive log directory using one of the following methods:

```
Repository>>setArchiveLogDirectory:
Repository>>setArchiveLogDirectory:tranlogPrefix:
Repository>>setArchiveLogDirectories:
Repository>>setArchiveLogDirectories:tranlogPrefix:
```

The `tranlogPrefix:` argument allows you to use a different setting for STN_TRAN_LOG_PREFIX on the production and standby systems.

Since restoring transaction logs terminates the session, you will need to login and set the archive logs for each restore. For example:

```
topaz> login
<details omitted>
successful login
topaz 1> printit
SystemRepository setArchiveLogDirectory 'GS-archive'.
System restoreFromArchiveLogs.
%
```

For more information about restoring logs, see page 274.

Step 7. Repeat Steps 5 and 6 as necessary.

You may find it necessary to shut down the standby from time to time. Ensure that you shut down the stone using stopstone. This does not affect the restore status.

Activate the warm standby in case of failure in the primary

Step 1. If the primary system fails, replay its latest transaction log on the standby system.

Step 2. On the standby server, send the message `Repository>>commitRestore` to terminate the restore process and enable logins.

Step 3. Client applications will have to reconnect to the standby system, which now becomes the primary system. Applications may have to perform their own failure recovery code as necessary, as well.

NOTE

Design your applications so that, after detecting a failure, they can determine which system is the new primary and reconnect correctly.

Step 4. Correct the problem on the failed system and restart it.

Depending on how much time has elapsed since the standby system became the primary system, either make a full backup of the new primary system and restore it on the system that failed, or replay the new primary system's transaction logs on the system that failed. Maintain that system in restore mode as the new standby.

Chapter

2 Configuring Gem Session Processes

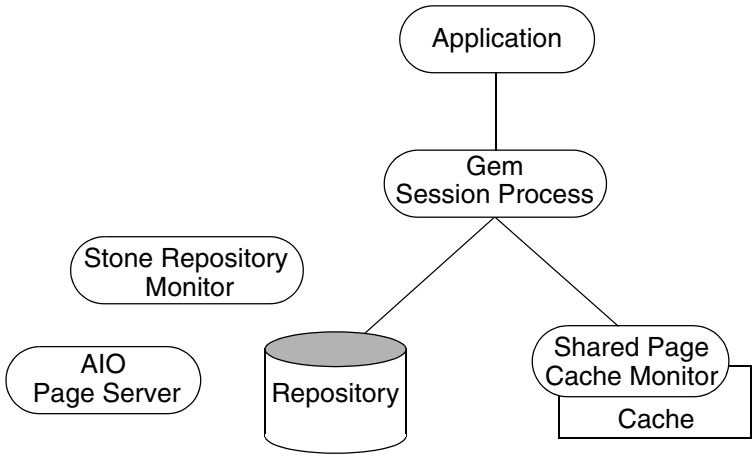
This chapter tells how to configure the GemStone/S 64 Bit session processes for your application. For additional information about running session processes on a node remote from the Stone repository monitor, refer also to Chapter 3.

2.1 Overview

As shown in Figure 2.1, a GemStone session involves the following components in a client-server relationship:

- The user application
- A session manager process (Gem), which acts as a server for a particular application
- The Stone repository monitor
- The shared page cache monitor and cache
- The Stone's AIO page server
- The repository itself

Figure 2.1 GemStone Session Elements



The Gem session process provides the bulk of the repository capabilities as seen by the application. From the viewpoint of the application, the Gem *is* the object server:

- It logs in to the repository through the Stone repository monitor, and it obtains object locks, free object identifiers, and free pages from the repository monitor.
- It presents the application with a consistent view of the repository during a transaction and tracks which objects the application accesses.
- It executes Smalltalk methods within the repository.
- It reads the repository as the application accesses objects, and (with the help of the AIO page server) it updates the repository when the application successfully commits a transaction.

Linked and RPC Applications

The Gem session process can be run as a separate process (as in Figure 2.1) or integrated with the application into a single process, in which case the application is called a *linked* application.

When the Gem runs as a separate process, it responds to Remote Procedure Calls (RPCs) from the application, in which case the application is called an *RPC* application. Applications that use a separate Gem process start that process automatically (from the user's viewpoint) while logging in to the repository.

GemStone provides both linked and RPC tools for repository administration. GemStone also provides both types of libraries for application developers. RPC applications start the Gem session process as part of connecting a user to the repository.

NOTE

Whether an application is linked or RPC depends on which GemStone library was loaded at run time. Either type of application can be used on a single node or across a network. Only one session can be linked, but the application can have multiple RPC sessions. C programmers should use an RPC version during development and debugging to protect Gem data structures from possible corruption.

The Session Configuration File

At start-up time, each Gem session process looks for a configuration file, which by default is the same system-wide configuration file sought by the repository monitor when it starts. However, there are three important differences:

- The session configuration file is optional. If one is not found, the session process uses system defaults.
- All session processes read those configuration options that begin with "GEM_" and the few that are used by both Stones and Gems (such as DUMP_OPTIONS and LOG_WARNINGS) . Other settings that the Gem needs are obtained from the repository monitor by network protocol and are the same for all sessions logged in to that Stone.
- The first session process on a node remote from the Stone and extents uses the shared page cache configuration options (SHR_), which determine the configuration of the cache on that node.

Sometimes it's useful for certain sessions to use a variant configuration. Appendix A, "GemStone Configuration Options," tells how to specify an alternate configuration file and how to use supplementary files to adjust the system-wide configuration for a specific session process. Appendix A describes each of the configuration options.

2.2 How To Configure Gem Session Processes

To configuring a Gem session process, you perform the following steps:

1. Gather application specifics about the number of sessions that will be logged in to the repository simultaneously from this node.
2. Plan the operating system resources that will be needed: memory and swap (paging) space.
3. Set the Gem configuration options.

   If this node is remote from the repository monitor, enable a local GemStone shared page cache. Gem session processes running on a server node always use the Stone's shared page cache.
4. Set GemStone file permissions to allow session processes access while providing adequate security.

Gathering Application Information

System resources needed for session processes primarily depend on the number of sessions that will be logged in to a particular repository from this node. Remember that in some applications each user can have more than one session logged in.

Planning Operating System Resources

GemStone session processes need adequate memory and swap space to run efficiently. In addition, kernel parameters can limit the number of sessions that can connect to the shared page cache.

Estimating Memory Needs

Two factors determine the memory needs for session processes:

- The size of the shared page cache on a node remote from the Stone and extents will depend on the configuration of the Gem that starts the cache. (There is only one cache on each node for a particular repository; session processes running on the server node attach to the Stone repository monitor's cache.)
- The amount of memory required by a Gem session is dependent on how it is configured, as determined by the system requirements. To avoid out-of-memory conditions, Gems must be configured with an adequate temporary object cache.

Assuming the default of 10 MB for the Gem's temporary object cache, the first Gem session process on a node ordinarily requires about 30 MB of memory, of which 5 MB is for code that can be shared by other session processes. Each additional session process requires about 25 MB. The requirement is the same for Gems linked with an application. If you tune the cache size for Gems (page 87), add any increase to the amount given here.

In addition to the memory needs for session processes, you must also allocate memory for object server processes. For details, see "Planning Operating System Resources" on page 35.

For Gem session processes running on machines that are remote from the object server, there are additional memory needs on the server. For information about this, see "Estimating Memory Needs" on page 35.

Estimating Swap Space Needs

Swap (paging) space on machines remote from the Stones should follow the same general guidelines given for servers on page 36. In determining the additional swap space needed for GemStone session processes, use the memory requirements derived in the preceding section ("Estimating Memory Needs"), including space for the number of sessions you expect. The resulting figures will approximate the client's needs, and are in addition to the swap requirement for the object server and non-GemStone processes.

Estimating File Descriptor Needs

When a Gem session process starts, it attempts to raise the file descriptor limit from the default (soft) limit to the hard limit set by the operating system. GemBuilder applications and page servers do the same. Gem session processes use file descriptors this way:

7 for stdin, stdout, stderr, and internal communication
2 for a connection between the Gem and an RPC application
1 for each local extent within a file system
2 for each local extent that is a raw partition
1 for each extent on a remote node

GemBuilder applications that start a large number of RPC Gems need a correspondingly large number of file descriptors.

You can override the default behavior of raising the file descriptor limit to the hard limit by setting the GEMSTONE_MAX_FD environment variable to a positive integer. A lower limit may be desirable in some cases to reduce the amount of

virtual memory used by the process. A value of 0 disables attempts to raise the default limit.

The value of GEMSTONE_MAX_FD in the environment of a NetLDI (Network Long Distance Information) server is passed to its child processes.

Reviewing Kernel Tunable Parameters

The kernel parameter of primary relevance to GemStone session processes is the maximum number of semaphores per semaphore id (typically `semmsl` or a similar name, although it is not tunable under all operating systems). This parameter limits the number of sessions than can connect to the shared page cache, because each session uses two semaphores.

How you determine the existing limits depends on your operating system. If the information is not readily available, proceed with setup. A later step shows how to verify that the limits are adequate for the GemStone configuration you set up.

Set the Gem Configuration Options

Initially, you should focus on configuration options for the shared page cache. Changes to other Gem configuration options are discussed later in this chapter, beginning on page 87.

The Stone repository monitor always creates a shared page cache monitor and cache on its node, based on parameters in the Stone's configuration file.

On other nodes, when the first Gem session process logs in, the Stone starts a shared page cache monitor and cache on that node. Parameters in the configuration file used by the first Gem that logs in determine the size of the cache and the number of processes that can attach to it (SHR_PAGE_CACHE_SIZE_KB and SHR_PAGE_CACHE_NUM_PROCS, respectively). All subsequent sessions that log in from that remote node will use the same cache.

You can use the GemStone **shmem** utility (described on page 42) to determine whether the kernel configuration on a node remote from the Stone is adequate to support the cache. Use arguments from the configuration file that will be read by Gems running on that node.

To Set Ownership and Permissions for Session Processes

The primary consideration in setting file ownership permissions for client access is to make sure the Gem session process can read and write both the extents and the shared page cache. You must also take into account the following factors:

- Is the Gem session process linked or RPC?
- Does the Gem session process runs on the server or on a node remote from the Stone.?
- Does the server uses setuid bit and protection mode 600 for the extents (as recommended on page 55), or does it use the alternative of group write permission?

Extents

The extents may be protected as read-write only by their owner (protection 600) if you use the setuid (S) bit for repository executables as recommended on page 55. Otherwise, the extents must be writable by a group to which the GemStone users belong (protection 660).

Shared Page Cache

The shared memory and semaphore resources used by GemStone are created and owned by the user account under which the Stone repository monitor is running and have the same group membership. Access is read-write for the owner and group (the equivalent of file protection 660). You can inspect the cache ownership and permissions by using the **ipcs** command. (These permissions are not configurable by users.)

For a session to log in using a shared page cache, the Operating System user account of the linked application or Gem session process must either be the same as that of the Stone (such as the *gsadmin* account) or be one that belongs to the same group as the Stone. The same requirement applies to page server processes, which are discussed in Chapter 3, "Connecting Distributed Systems."

If the setuid bit is set on repository executables as recommended in Table 1.2 on page 55, the Stone process and shared page cache will belong to the owner you specify for those files (such as *gsadmin*).

To Set Access for Linked Applications

For linked applications *on the server*, we recommend you try using the setuid bit on the application's executable file. Have the file owned by *gsadmin* as it is defined on page 55. This works well for **topaz -l**. The `installgs` script offers to set the file ownership and permissions for you. To do it manually, do this while logged in as root:

```
# cd $GEMSTONE/bin
# chmod u+s topaz
# chown gsadmin topaz
```

You may prefer not to use the setuid bit with linked applications that do not distinguish between real and effective user IDs. GemStone's Topaz executable performs repository reads and writes as the *effective* user (the account that owns the executable's file), but performs other reads and writes as the *real* user (the one who invoked it). Linked applications that do not make this distinction, such as a third-party Smalltalk used with GemBuilder, are likely to perform *all* I/O as the effective user, or *gsadmin.* If this result is unsatisfactory, remove the S bit on that executable and add group write permission to the extents.

To Set Access for All Other Applications

All applications except linked applications on the server require a GemStone NetLDI service to start a separate Gem session process or, in some cases, a page server. For these sessions, we recommend that the Gem session process and page server always be owned by (run as) the *gsadmin* account. That arrangement ensures that the Gem will be able to read and write both the extents and the shared page cache. The ownership and protection of the application executables themselves is not a factor.

To Set Access to Other Files

GemStone creates log files and other special files for session processes in several locations, which are described below. In a multi-user environment, the protection of these resources must be such that the appropriate file can be created or updated in response to actions by several users.

$HOME	GemStone ordinarily creates log files for spawned processes (such as RPC Gem session processes and page servers) in the home directory of the user or the NetLDI captive account. In situations where the home directory cannot be writable, the environment variable GEMSTONE_NRS_ALL can be used to specify an alternative location; see "To Set a Default NRS" on page 105.
`/opt/gemstone`	All users should have read/write/execute access to the directories `/opt/gemstone/log` and `/opt/gemstone/locks` on each host (or an equivalent location, as discussed on page 38). By default, NetLDI (Network Long Distance Information) processes create log files in the `log` directory. Repository-wide lock files are created in the `locks` directory, and other processes use this location to look up access information for them.

2.3 How To Access the Configuration at Run Time

GemStone provides several methods in class System that let you examine, and in certain cases modify, the session configuration parameters at run time.

To Access Current Settings at Run Time

Class methods in category Configuration File Access let you examine the configuration of your current Gem session process. There are three access methods for session processes:

`gemConfigurationReport`
: Returns a SymbolDictionary whose keys are the names of configuration file parameters, and whose values are the current settings of those parameters in the current session's Gem process.

`gemConfigurationAt:` *aName*
: Returns the value of the specified configuration parameter from the current session, or returns `nil` if that parameter is not applicable to a session process.

`configurationAt:` *aName*
: Returns the value of the specified configuration parameter, giving preference to the current session process if the parameter applies to a Gem.

To Change Settings at Run Time

The class method `System` class >>`configurationAt:` *aName* `put:` *aValue* in category Runtime Configuration Access lets you change the value of the internal run-time parameters in Table 2.1 if you have the appropriate privileges. The parameters that can be changed are those for which `configurationParameterDict:` *aName* returns a negative SmallInteger. All changeable parameters require that *aValue* be a SmallInteger.

CAUTION

Do not change configuration parameters unless there is a clear reason for doing so. Incorrect settings can have serious adverse effects on GemStone performance. Appendix A provides additional guidance about run-time changes to specific parameters.

Table 2.1 Session Configuration Parameters Changeable at Run Time

Configuration File Option	Internal Parameter
GEM_ABORT_MAX_CRS	#GemAbortMaxCrs
(none)	#GemConvertArrayBuilder
(none)	#GemDropCommittedExportedObjs
(none)	#GemExceptionSignalCapturesStack
GEM_FREE_FRAME_LIMIT	#GemFreeFrameLimit
GEM_FREE_PAGEIDS_CACHE	#GemFreePageIdsCache
GEM_KEEP_MIN_SOFTREFS	#GemKeepMinSoftRefs
GEM_NATIVE_CODE_ENABLED	#GemNativeCodeEnabled
GEM_PGSVR_COMPRESS_PAGE_TRANSFERS	#GemPgsvrCompressPageTransfers
GEM_PGSVR_UPDATE_CACHE_ON_READ	#GemPgsvrUpdateCacheOnRead
GEM_SOFTREF_CLEANUP_PERCENT_MEM	#GemSoftRefCleanupPercentMem
GEM_TEMPOBJ_POMGEN_PRUNE_ON_VOTE	#GemPomGenPruneOnVote
GEM_TEMPOBJ_POMGEN_SCAVENGE_INTERVAL	#GemTempObjPomgenScavengeInterval

The following example changes the value of the configuration option #GemFreeFrameLimit:

```
topaz 1> printit
System configurationAt:#GemFreeFrameLimit put: 4000
%
4000
```

For more information about the parameters that can be changed at run time, see Appendix A, "GemStone Configuration Options."

2.4 How To Tune Session Performance

There are a number of configuration options by which you can tune your Gem session processes. These options can help make better use of the Gem's internal caches, reduce swapping, and control disk activity limiting the I/O rate for certain sessions.

To Tune the Temporary Object Space

You should increase GEM_TEMPOBJ_CACHE_SIZE (page 369) for applications that create a large number of temporary objects — for example, applications that make heavy use of the reduced conflict classes or sessions performing a bulk load.

It is important to provide sufficient temporary object space. If temporary object memory is exhausted, the Gem can encounter an out-of-memory condition and terminate. This is particularly likely to be a problem if there are long transactions that modify a large number of objects.

You will probably need to experiment somewhat before you determine the optimum size of the temporary object space for the application. The default of 10000 (10 MB) should be adequate for normal user sessions. For sessions that place a high demand on the temporary object cache, such as upgrade, you may wish to use 100000 (i.e., 100 MB). Any increase in GEM_TEMPOBJ_CACHE_SIZE translates directly into increased memory usage per user.

Memory management is discussed in greater detail in Chapter 10, "Managing Memory", on page 295.

To Tune the Private Page Cache

The configuration option GEM_PRIVATE_PAGE_CACHE_KB sets the size (in KB) of the Gem's private page cache. The default value of this option is 1000; in most cases, this value is acceptable, and you do not need to do any further tuning.

You can use VSD to monitor the value of the statistic LocalCacheOverflowCount (page 492). It that value is non-zero, you can increase GEM_PRIVATE_PAGE_CACHE_KB as needed. For details, see page 367.

2.5 How To Install a Custom Gem

The *GemBuilder for C* manual explains how to create a custom Gem session executable containing your own C functions to be called from Smalltalk. One way to make this custom Gem available to all users is to perform the following steps as system administrator:

Step 1. Copy the shell script `gemnetobject` from `$GEMSTONE/sys` to your working directory. This shell script is used to start Gem session processes under the UNIX shell. You will modify this script to start your custom Gem executable instead of the standard one.

Step 2. In your copy of `gemnetobject`, find the section labeled `User-definable symbols`. In that section, replace `gem` in the line

```
gemname="gem"
```

with the name of the new Gem executable. For example:

```
gemname="MyGem"
```

Step 3. Rename your modified copy of the shell script `gemnetobject` so that it has a distinct filename. For example:

```
% mv gemnetobject MyGemnetObject
```

Step 4. Copy the new shell script to `$GEMSTONE/sys`. Make sure that all GemStone users have read and execute (**r-x**) permission for the script. For example:

```
-r-xr-xr-x 1 root 912 Jul 24 20:22 MyGemnetObject
```

If necessary, change the permissions:

```
% chmod 555 MyGemnetObject
```

Step 5. Add an entry for the new shell script to the services database, `$GEMSTONE/sys/services.dat`. A NetLDI checks that file to translate the name of a service to a command it can execute. For example:

```
MyGemnetObject $GEMSTONE/sys/MyGemnetObject
```

Step 6. Copy the new Gem executable to the GemStone system directory. For example:

```
% cp MyGem $GEMSTONE/sys
```

Step 7. Make sure that all GemStone users have read and execute (**r-x**) permission for the new Gem executable.

The custom Gem executable is now available for shared use.

Chapter

3

Connecting Distributed Systems

This chapter tells how to set up GemStone/S 64 Bit in a distributed environment:

- *Overview* (page 92) — An introduction to the GemStone processes and network objects that facilitate distributed GemStone systems.
- *How To Arrange Network Security* (page 99)— Three ways to provide access to GemStone processes on other nodes.
- *How To Use Network Resource Strings* (page 105) — How to specify where distributed GemStone resources are located.
- *How To Set Up a Remote Session* (page 106) — Step-by-step examples for setting up typical distributed client-server configurations. It also contains troubleshooting tips.

3.1 Overview

A properly configured network system is nearly transparent to GemStone users, but it requires additional steps by the system administrator. Users must be given access to all the workstations that will run their GemStone processes. Pointers to network services must be set up, and file and process specifications must include the node name in addition to the file name and path. Because processes are running on different nodes, the log files are spread throughout the network, and troubleshooting may become more complicated.

The nodes in your system can be any combination of GemStone-supported platforms, as long as they are connected by means of TCP/IP. Each remote GemStone connection consists of two TCP/IP connections to compensate for out-of-band problems in TCP/IP. (For remote Gems that are configured with GEM_PGSVR_UPDATE_CACHE_ON_READ set to true, only one TCP/IP connection is open.)

Although the Network File System (NFS) can be used to share executables, libraries, and configuration files, they are not required and are never used to share repository files. Instead, GemStone extends the capabilities of TCP/IP by adding special network servers and page servers, which are described later in this section.

Figure 3.1 and Figure 3.2 show two typical distributed configurations in which an application on a remote node is logged in to a repository and Stone repository monitor running on a server node.

In Figure 3.1, an application communicates with a Gem session process on the server node by way of RPC calls. This configuration lets the Gem execute Smalltalk code in the repository without first bringing complex objects across the network. The Gem can access the shared page cache that was started by the Stone repository monitor. For instructions on setting up this configuration, see “To Run the Gem Session Process on the Stone’s Node” on page 111.

In Figure 3.2, the application and the Gem are linked in a single process that runs on the remote node. This configuration avoids the overhead of RPC calls, but in some applications it may increase network traffic substantially if large objects must be brought across the network. The Stone repository monitor starts a shared page cache on the remote node when the first user from that node logs in to the repository. The Stone and the Gem session process each use a GemStone page server to access data pages residing on the other node. For instructions on setting up this configuration, see “To Run a Linked Application on a Remote Node” on page 108.

Figure 3.1 Gem Session Process on Server Node

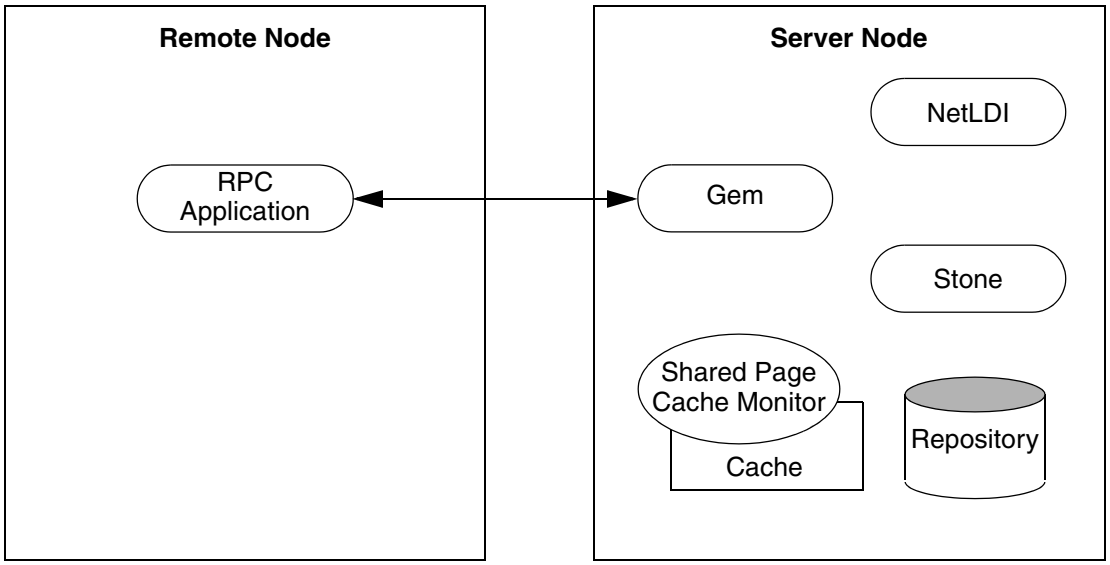
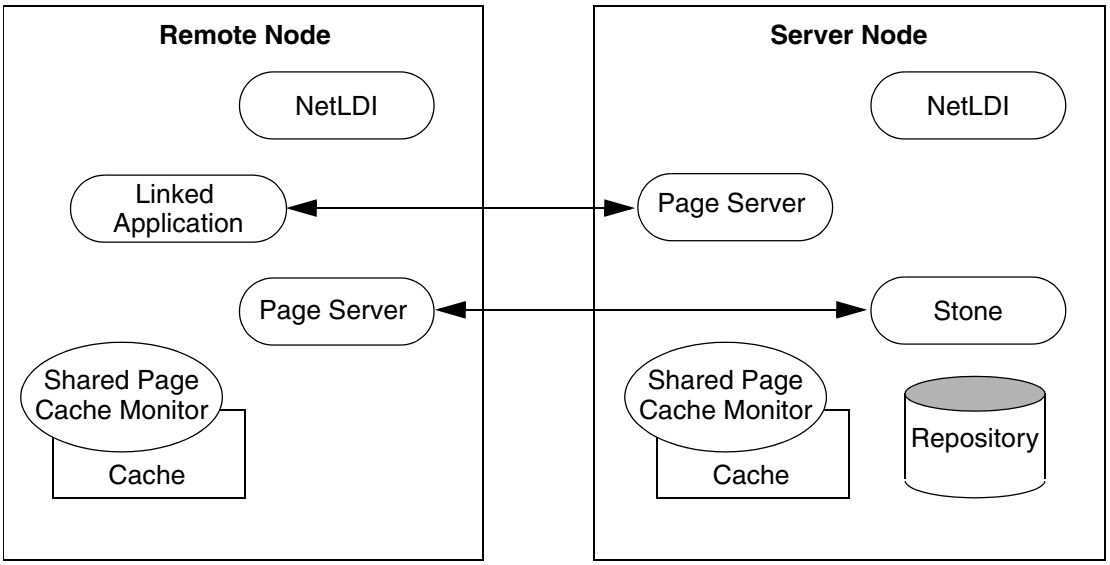


Figure 3.2 Gem Session Process on Remote Node



GemStone NetLDIs

The GemStone network server process is called NetLDI (Network Long Distance Information). The NetLDIs are the glue holding a distributed GemStone system together. Each NetLDI reports the location of GemStone services on its node to remote processes that must connect to those services. It also spawns other GemStone processes on request.

In a distributed system, each node where a Stone repository monitor, Gem session process, or linked application runs must have its own NetLDI. (That is, you do not need a NetLDI on nodes where only RPC applications are running.)

You start a NetLDI directly by invoking the **startnetldi** command (page 409). The NetLDI, in turn, starts Gem session processes and page servers on demand. (See the following section for more about page servers.) These child processes belong by default to the user account of the process requesting the service—sometimes that account is a user logging in to GemStone, other times it is the account that started the repository monitor.

Because most operating systems only let the root account start processes that will be owned by other accounts, a NetLDI may need to run as root if it is to serve more than one user. To accomplish this, you can either set the owner and S bit for `$GEMSTONE/sys/netldid` or, alternatively, start the NetLDI while logged in as root. You can also run netldi in captive account mode, in which all processes started by the NetLDI belong to a single user account. For more details on configuring NetLDI security, see "How To Arrange Network Security" on page 99.

NetLDI Names

The default name of the NetLDI process is `gs64ldi`. During installation, this name is added to the `/etc/services` file and is assigned a port number. You can specify a different name when you start the NetLDI: **`startnetldi`** *netLdiName*. If you use a different name, you should perform the following steps (which are required for some configurations):

- Add the new name and a port number to `/etc/services`. If you have a distributed GemStone system, make the same entry on each node.
- Set **#netldi** to *netLdiName* in the GEMSTONE_NRS_ALL environment variable for each user. For example:

```
$ GEMSTONE_NRS_ALL=#netldi:netLdiName
$ export GEMSTONE_NRS_ALL
```

For more information about GEMSTONE_NRS_ALL, see "To Set a Default NRS" on page 105.

You do not need to modify /etc/services, if you are willing to manage the NetLDI by port number, and you are not starting up remote shared caches.

NetLDI Ports

The NetLDI listens for all requests on a single, fixed TCP/IP port. This should normally be configured in the operating system file /etc/services. You may also set the NetLDI well-known port by using the **startnetldi -P** option, or by specifying a valid, unused port instead of a name in startnetldi.

Processes contact the NetLDI either by name or port.

- When contacting the NetLDI by name, the name is looked up in /etc/services. The NetLDI name must be mapped to the same port on every node that may be contacting the netldi, including client machines that do not need to run a NetLDI (such as client Windows machines running GemBuilder for Smalltalk).
- Alternatively, you can specify the NetLDI's port in any NRS used to contact a NetLDI, in which case you do not need to modify the /etc/services file.

The NetLDI uses a separate port to manage RPC logins. When a login request comes from a client, the NetLDI starts the server-side process and instructs it to listen on a port—not its well-known port—for the connection. The NetLDI then responds to the requestor with the port number to use when contacting the server-side process. The NetLDI normally selects an unused port for this purpose. To restrict the choice to a specified range of ports, as may be required by firewall configurations, use the **startnetldi -p** option, as shown here:

```
startnetldi -P 33333 -p44444:44446
```

For many systems, you may only need to specify a range of one port (e.g. -p 44444:44444). Exclusive use of this port is only required during the login itself. Once the connection is completed, the port may be reused for another login. Since the logins are serialized through these port or ports, heavily loaded systems may see delays during login if too few ports are specified.

Stone and Shared Page Cache Monitor

Every named GemStone process — Stone, NetLDI, and Shared Page Cache Monitor — creates a *serviceName*..LCK at startup, in the directory /opt/gemstone/locks or /usr/gemstone/locks, or in a directory specified by GEMSTONE_GLOBAL_DIR (as discussed on page 38). The .LCK file holds the well-known port that the process is listening on. When the NetLDI (for example)

contacts a particular Stone, it first looks up the well-known port for that Stone by locating the lock file for that Stone, then contacts the Stone on that port.

The ports used by the Stone and Shared Page Cache Monitor can be specified using the configuration parameters STN_WELL_KNOWN_PORT_NUMBER and SHR_WELL_KNOWN_PORT_NUMBER, respectively. These must be valid port numbers that are not already in use.

GemStone Page Servers

GemStone repository I/O is carried out by page server processes, running the executable file `pgsvrmain`. The Stone repository monitor creates one or more page servers at startup, to perform asynchronous I/O to the repository. There will also be Free Frame page servers that are dedicated to added free frames to the free frame list. For more about page servers, see page 68.

Sessions on remote hosts also require page servers. For each process that connects to a repository extent across the network, the NetLDI service spawns a `pgsvrmain` on the stone's node.

GemStone Network Objects

GemStone uses the concept of *network objects* to encompass the services that a NetLDI can provide to a client. In addition to the page server, other network objects include the following services requested by the Stone at startup: the shared page cache monitor, SymbolGem, page manager, and garbage collection (GcGem) sessions.

The network object most visible to users is the Gem session process requested by an RPC application. This object can be `gemnetobject` or the name of a custom Gem. The request can be sent to the NetLDI on the same node to start a local session process, or (by using a network resource string) to a NetLDI on another node to start a process there.

The NetLDI first tries to map the requested object to the path of an executable by looking for an entry in `$GEMSTONE/sys/services.dat.` That file contains an entry for the standard Gem session process:

```
gemnetobject              $GEMSTONE/sys/gemnetobject
```

For example, when you enter "gemnetobject" as a session login parameter (such as for gemnetid in Topaz), the NetLDI maps the request to the script `$GEMSTONE/sys/gemnetobject`. Similarly, an object name can be entered while setting up a GemBuilder session (as Name of Gem Service) or other

application. Application programmers provide the name as a parameter to `GciSetNet()`.

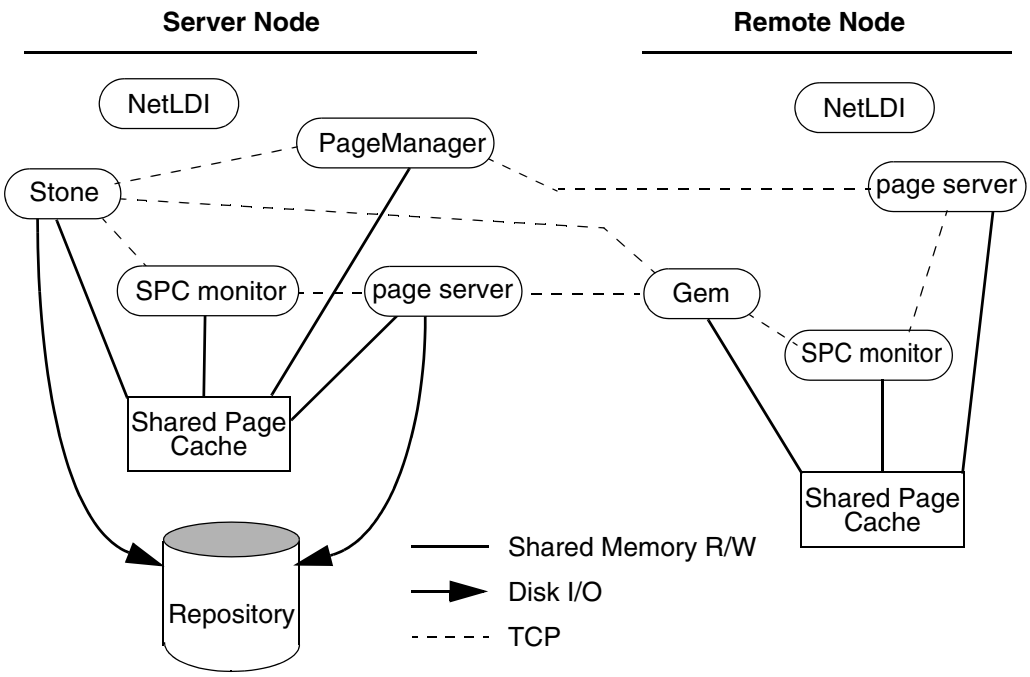
If your application uses a custom Gem executable, you can edit `services.dat` to include the appropriate mapping. For the procedure, see "How To Install a Custom Gem" on page 88.

If the NetLDI does not find the requested object in `services.dat`, it searches for an executable with that name in the user's $HOME directory. If you have a private Gem executable, place the executable in $HOME and then enter its name in place of *gemnetobject* during a GemStone login. Because of the search order, the private name must not be the same as that of an object in `services.dat`. The name must be the name of a file in $HOME, not a pathname.

Shared Page Cache in Distributed Systems

When the remote session logs in to the repository, the Stone repository monitor uses a NetLDI and page server (`pgsvrmain`) on the remote node to start a monitor process, and that monitor uses the NetLDI to create a local shared page cache. When the remote Gem wants to access a page in the repository, it first checks the shared page cache on the remote node. If the page is not found, the Gem uses a `pgsvrmain` on the server node, checking in the shared cache on that node and then, if necessary, reading the page from the disk. See Figure 3.3.

Figure 3.3 Shared Page Cache with Remote Gem



Disrupted Communications

Several incidents can disrupt communications between the GemStone server and remote nodes in a distributed configuration. This can include node crashes, firewalls, and loss of the communications channel itself.

GemStone ordinarily depends on the network protocol *keepalive* option to detect that a remote process no longer exists because of an unexpected event. The keepalive interval is set globally by the operating system, typically at two hours. When that interval expires, the GemStone process tries to obtain a response from its partner. The parameters governing these attempts also are set by the operating system, with up to 10 attempts in 15 minutes being typical. If no response is received, the local GemStone process acts as if its partner was terminated abnormally. In some cases, you may wish to adjust the keepalive interval to allow for a longer timeout.

3.2 How To Arrange Network Security

This section describes the levels to which the system administrator can set the GemStone authentication requirement.

Default

In the default NetLDI mode, authentication is required each time a NetLDI attempts to start certain processes for a client, even if that process is to run on the node where the user is logged in. These situations always require authentication:

- Starting an RPC Gem session process, even on the same node.
- Creating or restoring a GemStone backup using a device on a node remote from the Gem performing the operation.
- Using **copydbf** between nodes.

Once a Stone or Gem is running, the NetLDI treats it as a trusted client and starts the page servers needed by a remote login without authentication. Simple network information requests, such as a request to look up a port number, also do not require authentication.

Secure Mode

startnetldi -s starts the NetLDI in *secure mode*. All accesses are authenticated, including simple requests to look up a server name. This mode affects the **waitstone** command and such user actions as connecting a session process to a remote Stone (a NetLDI is asked to look up the Stone's address).

NOTE

Secure mode requires authentication before a Gem or Stone can start a page server to access an extent or shared page cache on another node. Under this mode, the account that starts the Stone process may need an entry in the account's .netrc file for each node in the GemStone system, and GemStone user accounts may need a .netrc entry for each node on which the extents are located.

Captive Account Mode

startnetldi -a*name* starts the NetLDI in *captive account mode,* which provides additional security. All child processes created by the NetLDI will belong to the single, designated account *name.* This mode requires that the NetLDI run either as root or as *name.* The effect is much like setting the S bits on executables, but it only affects ownership of processes started by the NetLDI, not linked applications

invoked directly by the user. Because this mode by itself does not change the authentication requirement, on most systems the NetLDI must either run as root so that it can authenticate other users or run in guest mode, which suspends authentication. For more information, see "Alternative: Guest Mode With a Captive Account" on page 104.

The captive account can be an ordinary user account or one created for that purpose, such as a GemStone administrative account. Log files by default will be in the captive account's home directory. Although captive accounts provide access to the repository, they do not affect network access—if authentication is required, it is based on the identity of the real user who requests the service.

Guest Mode

startnetldi -g starts the NetLDI in *guest mode*. No accesses are authenticated. When it is used by itself, guest mode lets the user who started the NetLDI also start other GemStone processes without typing passwords or creating `.netrc` files. Because guest mode is not permitted if the NetLDI will run as root, guest mode usually is combined with captive account mode. Table 3.1 shows how guest mode and captive account mode affect NetLDI operation.

Table 3.1 NetLDI Guest Mode and Captive Account Mode

NetLDI Options	Passwords Required	Owner of Spawned Processes	Owner of NetLDI Process	Which Accounts Can Start Processes
(none)	Yes, for RPC Gem or copydbf between nodes	Client's account	Ordinary user	Owner of NetLDI
			Root	Any user
-a*name*	Yes, for RPC Gem or copydbf between nodes	Account *name* (must start the NetLDI)	Ordinary user (*name*)	Owner of NetLDI
			Root	Any user
-g	No	Client's account	Ordinary user	Owner of NetLDI
			Root—not allowed	
-a*name* **-g**	No	Account *name* (must start the NetLDI)	Ordinary user (*name)*	Any user
			Root—not allowed	

For a complete list of the **startnetldi** command-line options, see page 409.

The following topics describe ways of setting up authentication to serve multiple users:

- Password authentication (the default) with the NetLDI running as root (page 102)
- Guest mode combined with a captive account (page 104)

Examples later in this chapter include procedures for specific configurations (see "Configuration Examples" on page 108).

Default: Password Authentication

The GemStone default is to use a system login name and password to authenticate network access. There are several ways for the user to provide this information:

- Create a `.netrc` file containing the name of the other node, the login name, and the password. (See "Using a .netrc File" on page 102.)
- Enter the login name and password through the application's user interface, such as the HostUserName and HostPassword parameters in Topaz. (See "Using the Application Interface" on page 103.)
- Use the NRS authorization modifier **#auth:***loginName@password* as part of a process name or file name. (See "Using an NRS #auth modifier" on page 103.)

If the user does not provide the login name and password explicitly, the application or GemStone executable tries to read them from a `.netrc` file in the user's home directory.

NOTE

Authentication is always done using the "real" user id, not the effective user id as set by the S bit on GemStone executables.

The NetLDI providing the service verifies the password against the entry in the password file (or Network Information Service). Under operating systems that support shadow password files, the NetLDI first checks the shadow file; if it finds an entry, it uses that entry in preference to the entry in `/etc/passwd`.

For the default installation, the file permissions and ownership for the NetLDI executable should look like this:

```
-r-sr-xr-x 1 root gsadmin 674488 Jun  7 11:29 netldid
```

Using a .netrc File

Create a `.netrc` file in the home directory of each user who will be doing any of the following:

- Running an RPC Gem session process.
- Creating or restoring a GemStone backup using a device on a node remote from the Gem performing the operation.
- Running **copydbf** between nodes.

If the user has a home directory on more than one node, the easiest way is to make a file containing an entry for each node and install a copy in all of the home

directories. The file must contain the login information for each node where that user will need an RPC Gem or a page server.

GemStone supports the basic `.netrc` options of `node`, `login`, and `password` (which must appear in that order). For each node, the `.netrc` file should contain one line like the following:

`machine` *nodeName* `login` *systemLogin* `password` *userPassword*

NOTE

In some cases, you may need to use the IP address, rather than the node name, in the `.netrc` *file.*

Because the `.netrc` contains hard-coded passwords, it should be protected in such a way as to be readable only by its owner.

Using the Application Interface

Your application's login interface may let you specify a node login name and password for the node on which you will be running an RPC Gem session process. For example, Topaz lets you set these as variables:

```
topaz> set hostusername yourLogin
topaz> set hostpassword yourPassword
```

GemBuilder for Smalltalk provides similar fields in its login dialog.

Using an NRS #auth modifier

A third way to specify the login name and password is to provide that information in NRS syntax as part of the name of a process that NetLDI is to start. Ordinarily, an application program provides the name and password using information obtained from the user. For example, if you set the Topaz login parameters *HostUserName* and *HostPassword*, the application puts them in an NRS like the following:

`'!@`*Server*`#auth:`*HostUserName*`@`*HostPassword*`!gemnetobject'`

The GemStone C interface provides similar capability to application programmers. For further information, refer to calls described in the *GemBuilder for C* manual.

Although it is less convenient for ordinary use, administrators and programmers may find it helpful in testing to enter the authorization modifier directly using the Topaz *GemNetId* parameter. For example:

```
topaz> set gemnetid !@Server#auth:name@password!gemnetobject
```

Alternative: Guest Mode With a Captive Account

The NetLDI guest mode can best be combined with captive account mode (page 99) in which a single, designated account owns all processes spawned by the NetLDI. The result serves multiple users with the convenience of guest mode and with improved security because the child processes no longer belong to accounts of individual users who request services.

The principal advantage of this combination is that the NetLDI can spawn processes on behalf of multiple users without being run as root. To make this capability possible, the captive account must own the `netldi` process. Change the file permissions and ownership for the NetLDI executable to remove the S bit:

```
-r-xr-xr-x 1 gsadmin gsadmin 674488 Jun  7 11:29 netldid
```

A disadvantage of the captive account for some applications is that the Gem session process will perform *all* I/O as that account, not as the account running the application — all file-ins, file-outs, and `System class >> performOnServer:`.

The captive account mode differs from the setuid method (page 55) in that captive account mode affects *all* services started by the NetLDI, including any ad hoc processes, which are processes started from the user's home directory. (The NetLDI looks in the user's home directory if it cannot find a service listed in `$GEMSTONE/sys/services.dat`.) If you prefer, you can prohibit such ad hoc services by specifying the **-n** option when starting the NetLDI.

If the combination of guest and captive account modes fits your needs, follow this configuration procedure:

Step 1. Create an OS account to own the GemStone distribution tree and serve as the captive account. We refer to this account as *gsadmin*.

Step 2. Make *gsadmin* the owner of the tree, and set the setuid bit for any linked GemStone executables that run on the server node. Make the repository extents accessible only by *gsadmin* (mode 600). For instructions, see "To Set File Permissions for the Server" on page 54.

Step 3. Make sure that *gsadmin* has execute permission for `$GEMSTONE/sys/netldid`. The setgid bit should NOT be set on the `netldid` executable (see above for how it should appear).

Step 4. Log in as the captive account (such as *gsadmin*). Then start the NetLDI in guest mode and captive account mode, and perhaps disallow ad hoc processes (the **-n** switch). For instance:

```
% startnetldi -g -a gsadmin -n
```

For details about the **startnetldi** command and its options, see page 409.

3.3 How To Use Network Resource Strings

Once you have chosen the remote and server nodes, network resource strings (NRS) allow you to specify the location of each part of the GemStone system. Use an NRS on a network system where you would use a process or file name on a single-node system. For example, suppose you want to know whether a Stone is running. If the Stone is on the local node, use this command:

```
% waitstone gemStoneName -1
```

If the Stone is on a remote node, use a command like this instead:

```
$ waitstone !@oboe!gemStoneName -1
```

where *oboe* is the Stone's node. You can also use an Internet address in "dot" form, such as `120.0.0.4`, to identify the remote node. Note that each "!" must be preceded by a backslash (\) when your command will be processed by the C shell.

Appendix B, "GemStone Utility Commands," indicates which options of each UNIX-level GemStone command can be specified as an NRS. Besides location, an NRS can describe the network resource type so that GemStone can more accurately interpret the command line. Sometimes an NRS can also include your authorization to use that resource. For more information, see Appendix C, "Network Resource String Syntax."

To Set a Default NRS

You can set a default NRS header (the part between "! ... !") by setting the environment variable GEMSTONE_NRS_ALL. This variable determines which modifiers GemStone will use by default in each NRS it processes on your behalf. For instance, you can cause all Gem session process logs to be created with a specific name in a specific directory.

- If you set GEMSTONE_NRS_ALL before starting a NetLDI, which is a system-wide service, that setting is passed to all its children and becomes the default for all users of that service.
- If you set GEMSTONE_NRS_ALL before starting a Stone, an application, or a utility (such as **copydbf**), that setting applies only to your own processes and does not affect other users.

Because these settings are defaults, they take effect only if an explicit setting is not provided for the same modifier in a specific request.

Use the **#dir** modifier to set the current (working) directory for NetLDI child processes, such as `gemnetobject`. Without this setting, the default is the user's home directory. If the directory specified does not exist or is not writable at run time, an error is generated. For example:

```
$ GEMSTONE_NRS_ALL=#dir:/user2/apps/logs
$ export GEMSTONE_NRS_ALL
```

For further information about the modifiers and templates available, see Appendix C, "Network Resource String Syntax."

3.4 How To Set Up a Remote Session

Configuring a Gem session process on a remote node is much the same as configuring a session process on the server, which is described in Chapter 2, "Configuring Gem Session Processes." Keep the following points in mind:

- A remote node (on which a Gem is running) must have its kernel configured for shared memory similarly to how it is configured on the primary server node.
- Only nodes running a Stone need a GemStone key file, not nodes running remote sessions.
- If your site doesn't run NIS, add each node in the GemStone network to `/etc/hosts`.
- If your site doesn't run NIS, add the NetLDI entry to `/etc/services` on each node. Be sure to specify the same name and network port number each time.
- It's best if each node has its own directory for `/opt/gemstone/log` and `/opt/gemstone/locks` (or `/usr/gemstone/log` and `/usr/gemstone/locks`, or other location under $GEMSTONE_GLOBAL_DIR). If these directories are on an NFS-mounted partition, make sure that two nodes are not using the same directories. Each Stone and NetLDI needs a unique lock file. Shared log files may make it impossible to diagnose problems.
- Unless you run the NetLDIs in guest mode with a captive account, users must have an account on, and authorized network access to, all nodes that are part of the GemStone network for the repository they will be using. This includes the nodes on which the Stone Repository monitor and the user's Gem session process reside.

- Unless you run the NetLDIs in guest mode with a captive account, *the user who starts the Stone repository monitor* ordinarily needs an account on *every* node where a Gem session process will run.

You can either repeat the installation from the GemStone distribution media, as described in the next topic, or mount the directory on the server node that contains $GEMSTONE (page 107). Although GemStone never uses NFS to access the repository files, it can use NFS to access other files.

To Duplicate the GemStone Installation

If you repeat the installation on the remote node, we recommend that you also run `$GEMSTONE/install/installgs`. In particular, you should make the same selections regarding the ownership and group for the GemStone files as you did on the primary server node. You can save disk space later by deleting initial repositories (`$GEMSTONE/data/extent0.dbf` and `$GEMSTONE/bin/*.dbf`) and the complete upgrade (`$GEMSTONE/upgrade`) and seaside (`$GEMSTONE/seaside`) directories.

To Share a GemStone Directory

The following example prepares to run an application and Gem session process on a remote node using a shared software directory on the server. The GEMSTONE environment variable points to the shared installation directory, which is on the node *Server* and is NFS-mounted as */Server*`/users/gs64stone`.

Step 1. Set the GEMSTONE environment variable to point to the NFS-mounted installation directory, and then invoke `gemsetup`:

(Bourne or Korn shell)

```
$ GEMSTONE=/Server/users/gs64stone
$ export GEMSTONE
$ . $GEMSTONE/bin/gemsetup.sh
```

(C shell)

```
Remote% setenv GEMSTONE /Server/users/gs64stone
Remote% source $GEMSTONE/bin/gemsetup.csh
```

Step 2. If they do not already exist, create the GemStone `log` and `locks` directories on the local node. (NetLDIs use this log directory.) You may need to have a system administrator do this for you as root.

```
# cd /opt
# mkdir gemstone gemstone/log gemstone/locks
# chmod 777 gemstone gemstone/log gemstone/locks
```

Configuration Examples

GemStone supports several configurations in which the application communicates with the Gem session process by using remote procedure calls (RPCs). Although the calls to network routines inevitably are time-consuming, they are essential when the application runs on a different node from the Gem, and they are desirable during code development because they isolate the application and Gem address spaces.

Use of RPC configurations for production repositories should be based on careful analysis of system loads and network traffic to select the most efficient configuration for a particular application. The RPC configuration may be desirable when the application accesses large or complex objects that would saturate the network if they were brought across it on a frequent basis.

This section presents examples that illustrate the following distributed applications:

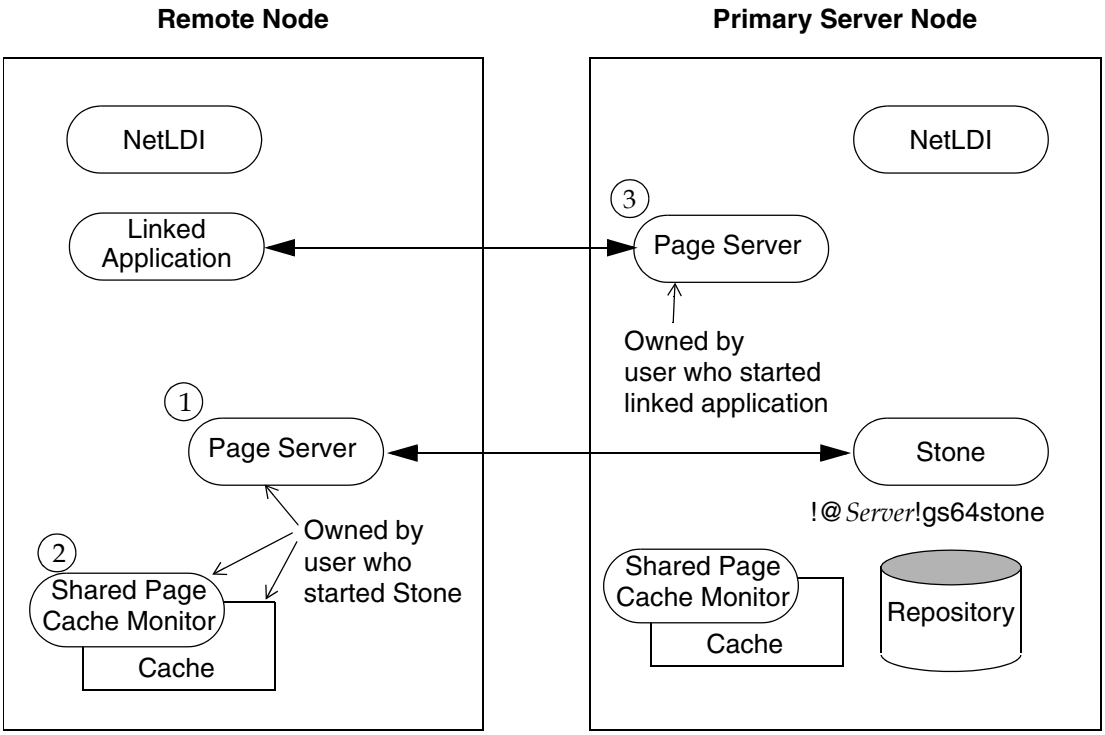
- A linked application connected to a Stone on another node (page 108).
- An RPC application with both the session process and the Stone on the server node (page 111).
- An RPC application with the session process on the application's node (page 113).
- An RPC application in which all three are on different nodes (page 116).

Two other examples show how to set up an extent on a node that is remote from the Stone, and how to use **copydbf** between nodes.

To Run a Linked Application on a Remote Node

Figure 3.4 shows how a linked application on a remote node communicates with a Stone and repository on the primary server node. This configuration typically is the best choice when you must offload some processes from a server node, especially when the application accesses relatively small objects or small groups of large objects.

Figure 3.4 Connecting a Linked Application to a Remote Server



Two NetLDIs and two page servers ordinarily are required. NetLDIs start the page servers on request of the Stone and the application. Numbers show the order in which these processes are started:

- One page server (1) lets the Stone start a shared page cache and monitor (2) on the remote node. The page server and monitor processes will be owned by the user who started the Stone (or by the captive account), so the owner must have an account on the remote node. The cache itself will have the same owner and group as the Stone. The linked application must have permission to access the cache, either through group membership or through an S bit on the application executable.
- The other page server (3) lets the Gem session process (the linked application) access the repository on the primary server. There will be one such page server

process on the primary server node for each session logged in from a remote node; its owner (which may be a captive account) must have an account on the primary server. The page server process must have read-write permission for the repository, either through group membership or through an S bit on the `pgsvrmain` executable.

Because the shared page cache is readable and writable only by its owner and members of the same group (protection 660), the user running the application may need to belong to that group. See "To Set Ownership and Permissions for Session Processes" on page 82.

The following steps set up a linked application on the remote node. They use software in an NFS-mounted installation on the primary server node. (That directory is already mounted on the remote node as */Server*/**gs64stone**.)

Step 1. Set the GEMSTONE environment variable to point to the installation directory, and then invoke `gemsetup`:

(Bourne or Korn shell)

```
$ GEMSTONE=/Server/gs64stone
$ export GEMSTONE
$ . $GEMSTONE/bin/gemsetup.sh
```

(C shell)

```
Remote% setenv GEMSTONE /Server/gs64stone
Remote% source $GEMSTONE/bin/gemsetup.csh
```

Step 2. Verify that a Stone and NetLDI are running on the primary server node. One way to do this verification is to use the **gslist** utility. For example:

```
Remote% gslist -m serverName
```

(The **-m** option tells **gslist** to list only processes that are running on the specified node. For more about **gslist**, see page 403.)

Step 3. Start a NetLDI on the remote node.

- To start the NetLDI for password authentication, make sure `$GEMSTONE/sys/netldid` is owned by root and has the S bit set. Issue this command (on some operating systems, you may have to issue it as root):

  ```
  Remote% startnetldi
  ```

- ❒ To start the NetLDI in guest mode (authentication is not required), make sure `$GEMSTONE/sys/netldid` does NOT have the S bit set. Log in as the captive account *name*, then issue this command:

  ```
  Remote% startnetldi -g -aname
  ```

Step 4. Start the linked application (for instance, Topaz) on the remote node, then set the *GemStone* login parameter to include the name of the primary server node in network resource syntax. For instance, to log in to Topaz as DataCurator:

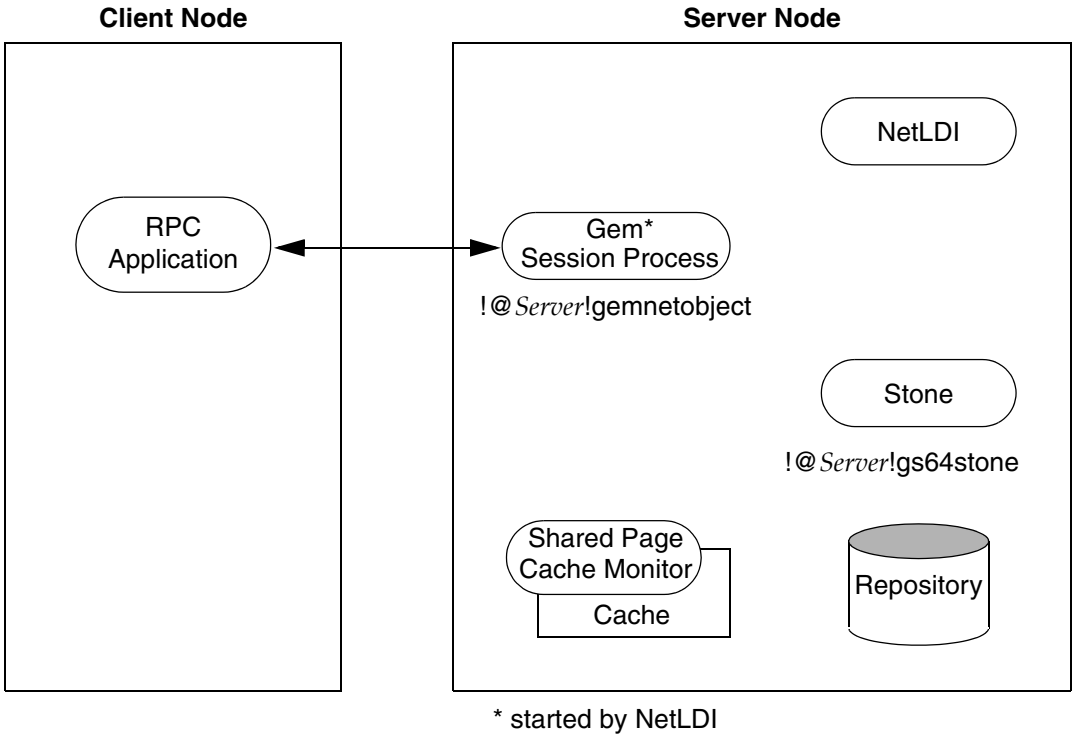
```
Remote% topaz -l
topaz> set gemstone !@Server!gs64stone
topaz> set username DataCurator
topaz> login
GemStone Password?
successful login
topaz 1>
```

To Run the Gem Session Process on the Stone's Node

If the Gem session process is going to run on the server node (as shown in Figure 3.5), an RPC application uses a NetLDI on that node to start a Gem session process. Unless the NetLDI is running in guest mode with a captive account, the application user must provide authentication to the NetLDI. You should also specify the Gem network object (`gemnetobject`) that matches your UNIX shell on the server. For more information about network objects and how to invoke them, see "GemStone Network Objects" on page 96.

The following procedure assumes that you are already set up to run GemStone applications, as described in Chapter 2. In particular, you must have defined the GEMSTONE environment variable and invoked `$GEMSTONE/bin/gemsetup.sh` or its equivalent.

Figure 3.5 Starting a Session Process on the Server Node



Step 1. Make sure that the NetLDI and Stone are running on the server. One way to do this is to use the **gslist** command. For example:

```
Client% gslist -m serverName
```

Step 2. Unless the NetLDI is running in guest mode, decide how you will provide authentication.

- You can create a `.netrc` file in your home directory on the client node containing a line like the following, where the password is your password on the server:

  `machine` *Server* `login` *yourLogin* `password` *yourPassword*

- Alternatively, you can set the application login parameters, such as HostUserName and HostPassword, after you start the application. For example:

  ```
  topaz> set hostusername yourLogin
  topaz> set hostpassword yourPassword
  ```

Step 3. Log in to the application node and start the RPC version of your application (for instance, Topaz), then set *UserName*. For example:

```
Client% topaz
topaz> set username DataCurator
```

Step 4. Set GemNetId to `gemnetobject`. Because the session process is to run on the server, be sure to include the node name in the GemNetId NRS. (It's not necessary to set the GemStone login parameter when the Stone repository monitor runs on the same node as the Gem.) For example:

```
topaz> set gemnetid !@Server!gemnetobject
```

Step 5. Log in to the repository:

```
topaz> login
GemStone Password?
successful login
topaz 1>
```

At this point, you are logged in to a Gem session process on the server node. That session process acts as a server to Topaz RPC and as a client to the Stone.

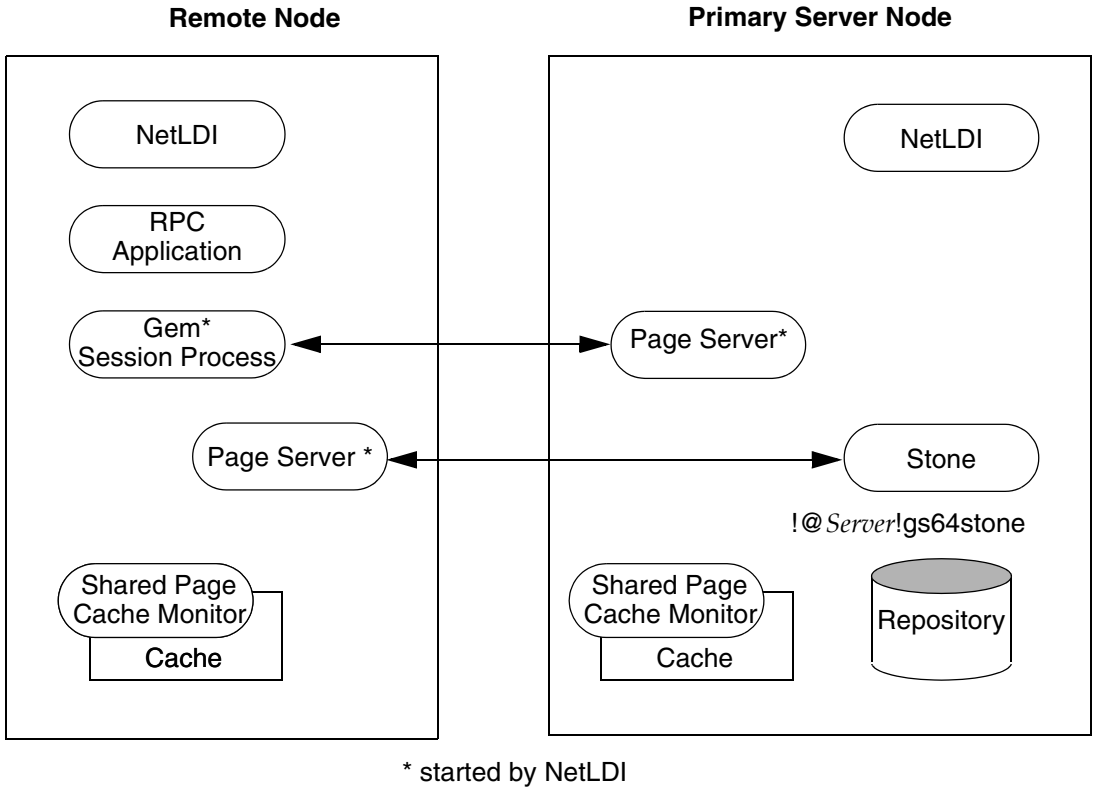
To Run the Gem and Stone on Different Nodes

The configuration shown in Figure 3.6 is unusual in that the RPC application and its session process are running on the same node. (While this configuration might be desirable during application development, a linked application, if it is available, probably would give better performance.)

The NetLDIs and page servers function similarly to those described for the linked application (see "To Run a Linked Application on a Remote Node" on page 108).

In Figure 3.6, however, the NetLDI also starts the RPC Gem session process at the request of the application.

Figure 3.6 Starting the Session Process on a Remote Node



Step 1. Unless the NetLDIs are running in guest mode, decide how you will provide access so that application can start a Gem session process on the remote node.

- You can create a `.netrc` file in the your home directory on the remote node containing a line like the following, where *userPassword* is your operating system password on the server:

```
machine remoteNode login userName password userPassword
```

- Alternatively, you can set the application login parameters, such as HostUserName and HostPassword, after you start the application. For example:

```
topaz> set hostusername yourLogin
topaz> set hostpassword yourPassword
```

Step 2. Log in to the remote node and start a NetLDI.

- To start the NetLDI for password authentication, make sure that `$GEMSTONE/sys/netldid` is owned by root and has the S bit set. Issue this command (on some operating systems, you may have to issue it as root):

```
Remote% startnetldi
```

- To start the NetLDI in guest mode (authentication is not required), make sure `$GEMSTONE/sys/netldid` does NOT have the S bit set. Log in as the captive account *name*, then issue this command:

```
Remote% startnetldi -g -aname
```

Step 3. Start the RPC version of your application (for instance, Topaz):

```
Remote% topaz
```

Step 4. Set GemNetId to `gemnetobject`. This network object identifies scripts that start a session process. For example:

```
topaz> set gemnetid gemnetobject
```

Step 5. Set the GemStone name, using NRS syntax to specify its location on the primary server node. Then set the UserName and log in. For example:

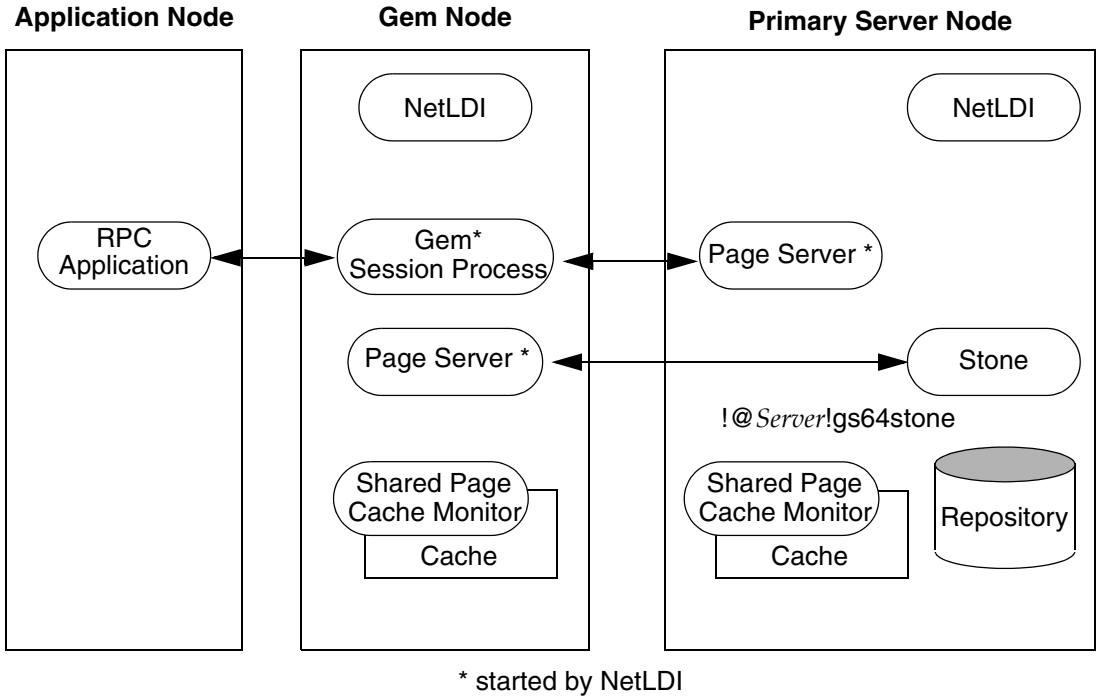
```
topaz> set gemstone !@Server!gs64stone
topaz> set username DataCurator
topaz> login
GemStone Password?
successful login
topaz 1>
```

To Run the Application, Gem, and Stone on Three Nodes

The RPC application, session process, and Stone can run on three separate nodes, as shown in Figure 3.7. The application runs on its node and connects to a Gem session process on the Gem's node. That session process communicates with the repository on the primary server node by way of a page server.

Again we see that a NetLDI must be running on each node where part of GemStone executes (but not necessarily on the application node, which runs only the RPC application).

Figure 3.7 Connecting an RPC Application, Three Nodes



The network access problem is similar to that in other RPC configurations: unless the NetLDI on the Gem node is running in guest mode, you must provide authentication to start the Gem session process.

Step 1. Unless the NetLDI on the Gem node is running in guest mode, decide how you will provide authorization for network services on that node.

- You can create a `.netrc` file in the your home directory on the application node containing a line like the following, where the password is your password on the Gem's node.

  `machine` *Gem* `login` *userLogin* `password` *secret2*

- You can set the application login parameters, such as HostUserName and HostPassword, after you start the application. For example:

  `topaz>` **`set hostusername`** *yourLogin*
  `topaz>` **`set hostpassword`** *yourPassword*

Step 2. Log in to the Gem's node and start the NetLDI.

- To start the NetLDI for password authentication, make sure that `$GEMSTONE/sys/netldid` is owned by root and has the S bit set. Issue this command (on some operating systems, you may have to issue it as root):

  `Remote%` **`startnetldi`**

- To start the NetLDI in guest mode (authentication is not required), make sure `$GEMSTONE/sys/netldid` does NOT have the S bit set. Log in as the captive account *name*, then issue this command:

  `Remote%` **`startnetldi -g -a`***name*

Step 3. Log in to the application node. Start the RPC version of your application (for instance, Topaz):

`Application%` **`topaz`**

Step 4. Set GemNetId to `gemnetobject`, and include the location, *gemNode,* in the NRS. For example:

`topaz> set gemnetid !@`*gemNode***`!gemnetobject`**

Step 5. Use NRS syntax to specify the location and name of the repository. Then set the username and log in. In Topaz, for example, set GemStone and UserName:

```
topaz> set gemstone !@Server!gs64stone
topaz> set username DataCurator
topaz> login
GemStone Password?
successful login
topaz 1>
```

At this point, your Topaz application on the application node has logged you in to a Gem session process on the Gem's node, and the session process has logged in to the repository on the server.

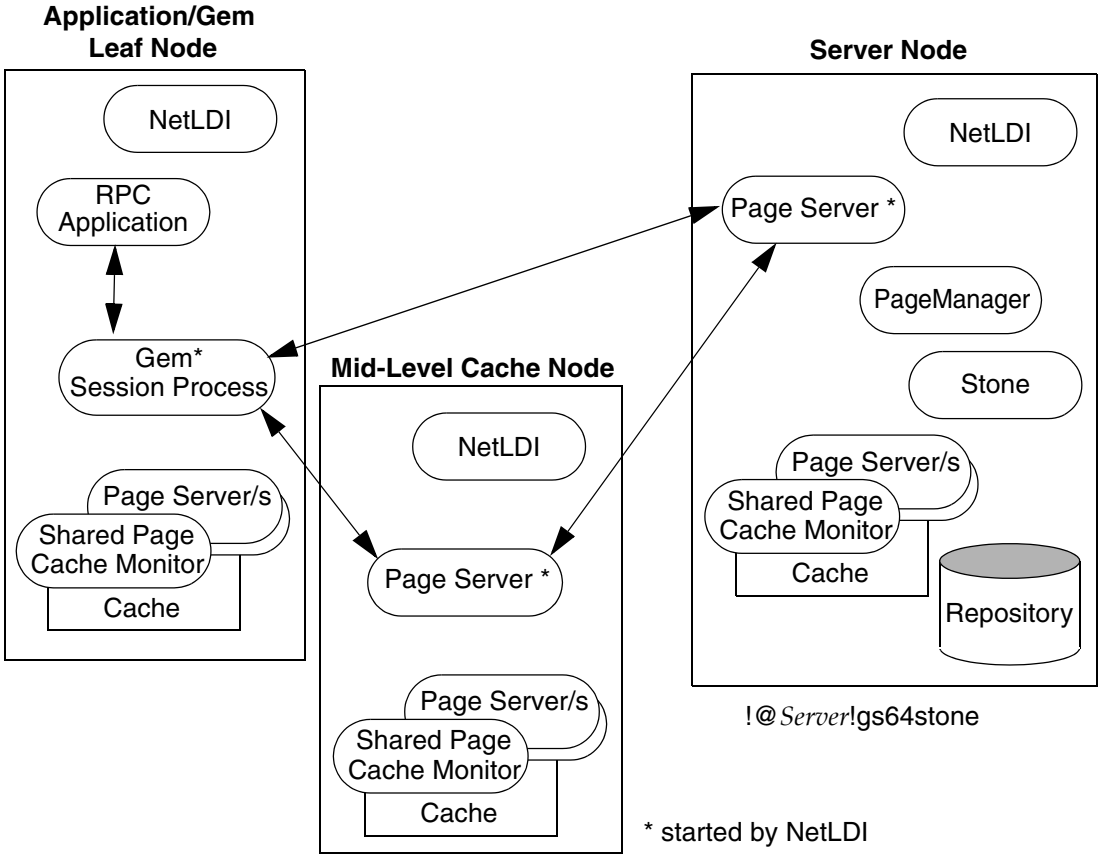
To Run a Distributed System with a Mid-Level Cache

In a distributed system over a Wide Area Network (WAN), with many remote nodes that are topographically distant from the Stone but close to each other, a mid-level cache can improve performance for the remote sessions. In this configuration, the RPC application and the session process may be on the same or different nodes, with the mid-level cache and Stone running on separate nodes. In Figure 3.8, the Gem session process (on the leaf node) connects to a mid-level cache (on the mid-level cache node). The session process communicates with the repository (on the primary server node) by way of a page server.

When the Gem needs a page but can't find it in its local cache, it first looks in the mid-level cache. If the Gem can't find the page in the mid-level cache, it then forwards the request to the page server on the Stone's host.

The page manager's pgsvr aggregates the responses from the pgsvrs on each of the gemSCs, and returns a combined response to the page manager. This reduces the number of round trips from the page manager to distant nodes.

If a mid-level cache is in use, then for each Gem process using the mid-level cache, all the shared caches to which the Gems are attached are subordinate to that mid-level cache.



Setting up a configuration with a mid-level cache requires that the session execute code following login to start a mid-level cache on a specified host, or to connect to an existing mid-level cache. Unlike the other remote configurations discussed in this chapter, the configuration is not established entirely by configuration settings and login arguments.

The Gem must be on a node that is remote from the Stone, and the request to connect to a mid-level cache must specify a node that is neither the Stone's nor the Gem's node.

If a Gem is running on the same machine as a mid-level cache, that Gem will use the mid-level cache as its local cache.

Connection Methods

System Class methods in the Shared Cache Management category allow you to connect to a mid-level cache.

`midLevelCacheConnect:` *hostName*

Attempts to connect to a mid-level cache on the specified host, if the cache already exists. The session's Gem process must be on a machine different from the machine running the Stone process.

`midLevelCacheConnect:` *hostName* `cacheSizeKB:` *aSize* `maxSessions:` *nSess*

If a mid-level cache does not already exist on the specified host, and *aSize* > 0, attempts to start the cache and connect to it. If a cache is already running on the host, this method attempts to connect to the cache and ignores the other arguments.

The size of the mid-level cache is controlled by the method argument *aSize*, rather than by configuration parameters (as with other shared caches).

Reporting Methods

System Class methods in the Shared Cache Management category return lists of the shared caches on your system.

`remoteCachesReport`

Returns a String that lists all shared caches that the Stone process is managing, not including the cache on the Stone machine.

`midLevelCachesReport`

Similar to `remoteCachesReport`, but only includes the mid-level caches.

For example,

```
topaz> set gemstone !@Server!gs64stone
topaz> set username DataCurator
topaz> set gemnetid !@gemNode!gemnetobject
topaz> login
GemStone Password?
<details omitted>
successful login
topaz 1>
topaz 1> printit
  System midLevelCacheConnect: 'midLevelCacheNode'
    cacheSizeKB: 2048 maxSessions: 10
%
```

3.5 Troubleshooting Remote Logins

Logging into GemStone from a remote node requires proper system configuration of the remote node and frequently requires permission for network access from the primary server to the remote node as well as from the remote node to the primary server.

- The UNIX kernel on the remote node should meet shared memory and semaphore requirements similar to those for the server, although smaller sizes may be sufficient.
- Make sure that NetLDIs are running on all nodes that require them (see the figure for your configuration). Also make sure that the NetLDIs have the same port number in `/etc/services`. All nodes must be listed in `/etc/hosts`.
- If an RPC application is being started (that is, one with a separate Gem session process), make sure the user who starts the application has an entry for the Gem's node in a `.netrc` file in $HOME, or that other authentication provisions have been taken, such as running the NetLDI in guest mode with a captive account. The owner of the Gem process needs an account on the node where the Gem will run and needs write access to the Gem log, typically in $HOME. Ownership and permissions for $GEMSTONE/sys/netldid must be appropriate for the authentication system in use (see pages 102 and 104), and the directories in `/opt/gemstone` must be writable.
- Make sure that the user who started the Stone has an account on the remote node. This user also must have write permission for $HOME so that log files

for the remote node can be created, unless steps are taken to create the log files in another directory.

- Check any GEMSTONE environment variables for definitions that point to a previous version: **env | grep GEM**.

If You Still Have Trouble

If you still can't log in to GemStone from an application on a remote node, try logging in on the server node as the same UNIX user account. We suggest that you first try a linked application, such as **topaz** −**l**, and when that works, move on to an RPC application (such as **topaz** or the equivalent **topaz** −**r**), still on the server.

Try Linked Topaz on the Server

A linked application on the server offers the least complicated kind of login because the server's shared page cache is already running and no network facilities are used. Any problems are likely to involve access permission for the shared page cache or the repository extents, which can also block attempts to log in from a client node.

- Make sure that the owner of the topaz process (`$GEMSTONE/bin/topaz`) can access the shared page cache. Use the UNIX command **ipcs -m** to display permissions, owner, and group for shared memory; for example:

```
Server% ipcs -m
IPC status from <running system> as of Mon Mar 21
16:22:27 PDT 2011
T     ID     KEY        MODE       OWNER    GROUP
Shared Memory:
m    768 0x4c177155 --rw-rw----  gsadmin     pubs
```

  Compare the owner and group returned by **ipcs** with the owner of the Topaz process. You can use the **ps** command to determine the owner; for example, **ps -ef | grep topaz**. (The switches may be different on your system.)

  A typical problem arises when root owns the Stone process and the shared page cache because their group ordinarily will be a special one to which Topaz users do not belong. Related problems may occur with a linked GemBuilder session. The third-party Smalltalk may be installed without the S bits and therefore may rely on group access to the shared page cache and repository. For background information, see "To Set Ownership and Permissions for Session Processes" on page 82.

  To correct a shared page cache access failure, either change the owner and group of the setuid files or have the Stone started by a user whose primary

group is one to which other GemStone users belong. Unlike file permissions, the shared page cache permissions cannot be set directly.

- Make sure the owner of the Topaz process has read-write access to `$GEMSTONE/data/extent0.dbf`.

Try Topaz RPC on the Server

The next step should be to try running Topaz on the server with a separate Gem session process. This configuration relies on the NetLDI to start a Gem session process, and that process, not the application itself, must be able to access the shared page cache and repository extent.

- Make sure that a NetLDI is running on the server by invoking **gslist**. The default name is `gs64ldi`. If you need to start a NetLDI, the command is **startnetldi**.

  GemStone uses the NetLDI to start a Gem session process that does repository I/O in this configuration. For the NetLDI to start processes for anyone other than its owner, it must be owned by root or it must be started in guest mode and captive account mode by someone logged in as the captive account.

  The NetLDI writes a log file with the default name `/opt/gemstone/log/gs64ldi.log`. The log file contents may help you diagnose problems. (See the following discussion, "Check NetLDI Log Files.")

- Make sure that the owner of the resulting Gem session process (`$GEMSTONE/sys/gem`) can access the shared page cache and `extent0.dbf` through group membership or S bits. The troubleshooting is the same as that given on page 123 for the `topaz` executable.

The user who starts topaz (or the NetLDI captive account when it is in use) must have write permission for $HOME so that the session process can create a log file there. (For a workaround for situations where write permission is not allowed, see "To Set a Default NRS" on page 105.)

Check NetLDI Log Files

Troubleshooting on a distributed GemStone system can be complicated. What looks like a hung process may actually be caused by incorrect NRS syntax or by another node on the network going down. The information for analyzing problems may be found in log files on all the nodes used by GemStone.

Where the log file messages include NRS strings, be sure to check their syntax. The problem may be as simple as an incorrect NRS or one that was not expanded by the shell as you intended.

If you can't identify the problem from the standard log messages, try running the NetLDI in debug mode, which puts additional information in the log. The command line is **startnetldi** [*netLdiName*] **-d**.

The following example shows how you might use the NetLDI log to diagnose problems during an RPC login.

1. The client (Topaz, GBS, or GCI) contacts the NetLDI and requests a session.

```
Entering Service Loop
0: --- 03/21/11 12:40:11.406 PDT :
  Attempting accept...
     ...succeeded accepting client from 10.80.8.12,
     connection = 2
0: --- 03/21/11 12:40:11.484 PDT :
  Finished reading client request:
     Client is a rpc application.
     '!#encrypted:<username>@password!gemnetobject'
```

   If there is no message in the NetLDI log after a login attempt, the failure is in connecting to the NetLDI. (This is usually clear from the error message.) Check your login parameters, particularly the NetLDI name, the port id assigned in the `/etc/services` file, and the GEMSTONE_NRS_ALL setting.

   If you started NetLDI using the **-p** option to specify a port range, verify that these ports are appropriate and, if you have a firewall, that these ports are open.

2. NetLDI starts listening on a "callback" port. This is not the port that the NetLDI uses to accept connections. This port is just used temporarily for the Gem-NetLDI communication.

3. NetLDI launches `gemnetobject` (or a similar script specified in the login parameters), telling it the number of the callback port it should use.

```
0: --- 03/21/11 12:40:11.499 PDT :
  Successful fork; Child's Pid: 28836 command is:
     '/<$GEMSTONE>/sys/gemnetobject TCP 48273 30'
```

   Note that "Successful fork" does not necessarily mean that the Gem was correctly started and initialized; the NetLDI does not block.

4. The `gemnetobject` script runs. If you are not using the default gemnetobject, and you have errors in your script, it is very difficult to diagnose them; since the problem is outside of the NetLDI or Gem, details are not reported in either the NetLDI or the Gem logs.

5. The `gemnetobject` script exec's gem to start running the Gem's code.
6. The Gem session initializes itself. At this point there is a Gem log. The pid of the gem process is shown in the NetLDI log (step 3).
7. The Gem starts listening on a client service port. This is yet another distinct port, and is the one that will be used for ongoing communication between the Gem and the client. The port id is shown in the NetLDI log (step 10).
8. The Gem connects to the NetLDI on the callback port.
9. The Gem tells the NetLDI that it is ready, and the number of its client service port:

   ```
   0: --- 03/21/11 12:40:11.803 PDT :
        Now reading reply from child
   ```

   If this step does not occur within the NetLDI timeout, the NetLDI times that Gem out and stops listening on the callback port. (The NetLDI log will contain a message to that effect.) If you are getting timeouts with the default timeout interval, try increasing the timeout by invoking **startnetldi** with the **-t***timeout* option. This may at least help you keep working while determining why the timeout is occurring.
10. NetLDI passes the number of the client service port back to the client (Topaz, GBS or GCI), informing it that a Gem has been started for it, and that this Gem is waiting for a connection at a specific port.

    ```
    0: --- 03/21/11 12:40:11.804 PDT :
      Reply to client started:
      'SUCCESS 48274'
    0: --- 03/21/11 12:40:11.804 PDT :
      Done writing reply to client.
    0: --- 03/21/11 12:40:11.885 PDT :
      Disposed. elapsed time = 0
    ```
11. The client contacts the Gem directly on its client service port.

Chapter

4

Running GemStone

This chapter shows you how to perform some common GemStone/S 64 Bit system operations:

- Starting the GemStone Object Server (page 128)
- Starting Network Long Distance Information (NetLDI) servers (page 134)
- Identifying running servers (page 136)
- Logging in to a GemStone session (page 136)
- Identifying the current sessions (page 143)
- Shutting down the object server (page 144)

Additional topics explain how to:

- Recover from an unexpected shutdown (page 146)
- Load objects in bulk (page 149)
- Enter and use manual transaction mode, which we recommend that you use as often as possible (page 149)

4.1 How To Start the GemStone Server

In order to start a Stone repository monitor, the following must be identified through your operating system environment:

- Where GemStone is installed

  The GEMSTONE environment variable must point to the directory where GemStone is installed, such as `/users/gemstone`. The directory `$GEMSTONE/bin` should be in your search path for commands.

- Which configuration parameters to use

  The repository monitor must find a configuration file. The default is `$GEMSTONE/data/system.conf`. Other files can supplement or replace the default file; for information, see "How GemStone Uses Configuration Files" on page 350.

- Which repository to use

  The configuration file must give the path to one or more repository files (extents) and to space for transaction logs. The default configuration file specifies `$GEMSTONE/data/extent0.dbf` for the extent file, and places transaction logs in `$GEMSTONE/data/`. You may want to move these files to other locations. For further information, see "Choosing the Extent Location" on page 44.

To Start GemStone

Follow these steps to start GemStone following installation or an orderly shutdown. (To recover from an abnormal shutdown, refer to "How To Recover from an Unexpected Shutdown" on page 146.)

Step 1. Set the GEMSTONE environment variable to the *full pathname* (starting with a slash) of the directory where GemStone is installed. Ordinarily this directory has a name like `GemStone64Bit3.0.0-sparc.Solaris`. For example:

```
$ GEMSTONE=/users/GemStone64Bit3.0.0-sparc.Solaris
$ export GEMSTONE
```

If you have been using another version on GemStone, be sure you update or unset previous settings of these environment variables:

- GEMSTONE
- GEMSTONE_SYS_CONF
- GEMSTONE_EXE_CONF
- GEMSTONE_LANG

Step 2. Set your UNIX path. One way to do this is to use one of the `gemsetup` scripts. There is one version for users of the Bourne and Korn shells and another for users of the C shell. These scripts also set your man page path to include the GemStone man pages.

(Bourne or Korn shell)

```
$ . $GEMSTONE/bin/gemsetup.sh
```

or (C shell)

```
% source $GEMSTONE/bin/gemsetup.csh
```

Step 3. Start GemStone by using the **startstone** command:

```
% startstone [gemStoneName]
```

where *gemStoneName* is optional and is the name you want the repository monitor to have. The default name is `gs64stone`. For additional information about **startstone**, see page 411.

To Troubleshoot Stone Startup Failures

If the Stone repository monitor fails to start in response to a **startstone** command, it's likely that the cause is one of the following. Inspect the Stone log for clues (the default location is `$GEMSTONE/data/gs64stone.log`), then refer to the discussions that follow this summary.

- The GemStone key file is missing or invalid (see page 130).
- The shared page cache cannot be attached (see page 130).
- An extent file is missing or cannot be opened for exclusive use because another GemStone process is using it (see page 131).
- Because of the timing of a system crash, the repository monitor is trying to create an extent that already exists (see page 132).
- A transaction log needed for recovery is missing, or the log directory or device does not exist (see page 133).

- The repository has become corrupted (see page 133).

Missing or Invalid Key File

The Stone repository monitor must be able to read the key file `$GEMSTONE/sys/gemstone.key`. Ordinarily, you create this file during installation from information provided by GemStone. Be careful to enter the information correctly. GemStone key files are platform-specific, and keyfiles for earlier versions may not work with new releases.

If you do not have a valid keyfile, contact GemStone Technical Support as described on page 7.

Shared Page Cache Cannot Be Attached

The shared page cache monitor must be able to create and attach to the shared memory segment that will serve as the shared page cache. Several factors may prevent this from happening:

- On some platforms, shared memory is not enabled in the kernel by default, or its default maximum size is too small to accommodate the GemStone configuration. GemStone's default configuration requires a shared memory segment slightly larger than 75 MB. For specifics about configuring shared memory, refer to the *GemStone/S 64 Bit Installation Guide* for your platform.
- If the size of the shared page cache has been increased, the operating system's limit on shared memory regions may need to be increased accordingly. Page 41 describes a utility (`$GEMSTONE/install/shmem`) that will help you check the configuration.
- The repository executables (the Stone, Gems, and page servers) must have permission to read and write the shared page cache. Ways to set up access are described in "To Set File Permissions for the Server" on page 54. In general, users must belong to the same group as the Stone repository monitor. If the Stone is running as root, it is unlikely that other users will be able to access the shared page cache.

Extent Missing or Access Denied

If the Stone repository monitor cannot access a repository extent file, it logs a message like the following:

```
GemStone is unable to open the file
!TCP@pelican#dbf!/users/GemStone/data/extent0.dbf.
reason = File = /users/GemStone/data/extent0.dbf
DBF Op: Open; DBF Record: -1;
Error: open() failure; System Codes: errno=2, ENOENT, The file or
directory specified cannot be found

An error occurred opening the repository for exclusive access.

Stone startup has failed.
```

Examine the message for further clues. The extent file could be missing, the permissions on the file or directory could be set incorrectly, or there may be an error in the configuration file that points to the extents. Correct the problem, then try starting GemStone again.

Extent Open by Another Process

If another process has an extent file open when you attempt to restart GemStone, a message like the following appears in the Stone log (by default, `$GEMSTONE/data/gs64stone.log`):

```
GemStone is unable to open the file
!TCP@pelican#dbf!/users/GemStone/data/extent0.dbf.
reason = File = /users/GemStone/data/extent0.dbf
DBF Op: Open; DBF Record: -1;
Error: exclusive open:  File is open by another process.; System
Codes: errno=11, EAGAIN, No more processes (due to process table
full, user quotas, or insufficient memory)

An error occurred opening the repository for exclusive access.

Stone startup has failed.
```

Close any other Gem sessions (including Topaz sessions) that are accessing the repository you are trying to restart, or wait for a **copydbf** to complete. Use **`ps -ef`** (the options on your system may differ) to identify any `pgsvrmain` processes that are still running, and then use **`kill`** *processid* to terminate them. Try again to start GemStone.

Extent Already Exists

If GemStone attempts to recover from a system crash that occurred just after an extent was created, and GemStone was not able to write a checkpoint when the extent was added, you will find an error message like the following in the Stone log:

```
An error occurred in recovery for extentId 2:
fileName= !TCP@pelican#dbf!/users/GemStone/data/extent1.dbf
File already exists; you must delete it before recovery can succeed.
```

Check that an extent was being added to the repository at or shortly before the crash. If necessary, look for a message near the end of the Stone log file.

- If an extent was being added, there is no committed data in the extent file yet. Delete the specified file and do not replace it with anything. Try to start GemStone again. The recovery procedure will recreate the extent file.
- If an extent was NOT being added, it is possible that an existing extent has been corrupted. For instance, `extent0.dbf` of a multiple-extent repository may have been overwritten. Try to determine the cause and whether the action can be rectified. You may have to restore the repository from a backup.

Other Extent Failures

At startup, the GemStone system performs consistency checks on each extent listed in DBF_EXTENT_NAMES.

All extents must have been shut down cleanly with a repository checkpoint the last time the system was run. This consistency check is the only one for which GemStone attempts automatic recovery.

The following consistency checks, if failed, cause the startup sequence to terminate. These failures imply corruption of the disk or file system, or that the extents were modified at the operating system level (such as by **cp** or **copydbf**) outside of GemStone's control and in a manner that has corrupted the repository.

- Extents must be in proper sequence within DBF_EXTENT_NAMES.
- Extents must be properly sequenced in time.
- The last checkpoint must have occurred earlier than or at the same time as the current system time (in GMT).
- Extents must belong to the correct repository.

Transaction Log Missing

If GemStone cannot find the transaction log file for the period between the last checkpoint and an unexpected shutdown, it puts a message like this in the Stone log:

```
Extent 0 was not cleanly shutdown; recovery is needed.
<Repository startup statistics>

Repository startup is from checkpoint = (fileId 6,
blockId 3)

ERROR: cannot find log file(s) to recover repository.
To proceed without tranlogs and lose transactions
committed since the last checkpoint use "-N" switch on
your startstone command.

An error occurred when attempting to start repository
recovery.
Waiting for aiowrites to complete

Stone startup has failed.
```

If the log file was archived and removed from the log directory, restore the file.

If the log file is no longer available, you can use **startstone -N** to restart from the most recent checkpoint in the repository. However, any transactions that occurred during the intervening period cannot be recovered.

NOTE

*When you use **startstone** with the **-N** option, any transactions occurring after the last checkpoint are permanently lost.*

Repository Failure

If you have GemStone backups (either online extent backups or Smalltalk full backups, as described in Chapter 9), you can restore the repository to the state of the most recent backup. If full logging (STN_TRAN_FULL_LOGGING = TRUE) was in effect at the time of the backup, objects committed by subsequent transactions can then be recovered from the transaction logs. For details, see "How To Restore from an Online Extent Backup" (page 263) or "How To Restore from a Smalltalk Full Backup" (page 269).

If you do not have a recent backup and transaction logs for valuable data, depending on the nature of the problem, in some case you may still be able to

recover your committed repository. See "How To Audit the Repository" on page 165.

Other Startup Failures

- Check `/opt/gemstone/locks` (or equivalent location, as discussed on page 38) and remove old files. On Solaris systems, also check `/tmp/gemstone` for *stoneName*`..FIFO`.
- Certain unexpected shutdowns may leave UNIX interprocess communication facilities allocated, which can block attempts to restart the repository monitor. Use the command **ipcs** to identify the shared memory segments and semaphores allocated, then use **ipcrm** to free those resources allocated to a repository monitor that is no longer running. For information about **ipcs** and **ipcrm**, consult your operating system's documentation.
- If it takes more than 5 minutes for your cache to complete initialization, the startup timeout may be expiring. Set the environment variable $GEMSTONE_SPCMON_STARTUP_TIMEOUT; see page 439.
- If you can't start GemStone under any circumstances, try **pageaudit** on the repository. (See "How To Audit the Repository" on page 165.) If the page audit is good but GemStone still doesn't start, check your installation configuration. For more help, contact your local GemStone administrator or GemStone Technical Support.

4.2 How To Start a NetLDI

It's common practice to start a GemStone Network Long Distance Information (NetLDI) server when starting a Stone repository monitor. Chapter 3 of this manual describes the two situations in which a NetLDI is necessary:

- A user will be running an RPC application with a separate Gem session process on the Stone's node.
- A user will be running a linked application or a separate Gem on another node and logging in to the repository on the Stone's node.

To start a NetLDI server, perform the following steps on the node where the NetLDI is to run:

Step 1. Set the GEMSTONE environment variable to the *full pathname* (starting with a slash) of the directory where GemStone is installed. Ordinarily this

directory has a name like `GemStone64Bit3.0.0-sparc.Solaris`.For example:

```
$ GEMSTONE=/installDir/GemStone64Bit3.0.0-sparc.Solaris
$ export GEMSTONE
```

Step 2. Use one of the `gemsetup` scripts to set your UNIX path. There is one version for users of the Bourne and Korn shells and another for the C shell. These scripts also set your man page path to include the GemStone man pages.

(Bourne or Korn shell)

```
$ . $GEMSTONE/bin/gemsetup.sh
```

or (C shell)

```
% source $GEMSTONE/bin/gemsetup.csh
```

Step 3. Start the NetLDI by using the **startnetldi** command.

- ❒ To start the NetLDI for password authentication, make sure that `$GEMSTONE/sys/netldid` is owned by root and has the S bit set. Issue this command (on some operating systems, you may have to issue it as root):

  ```
  % startnetldi
  ```

- ❒ To start the NetLDI in guest mode (authentication is not required), make sure that `$GEMSTONE/sys/netldid` does NOT have the S bit set. Log in as the captive account *name*, then issue this command:

  ```
  % startnetldi -g -aname
  ```

For additional information about **startnetldi**, see the command description in Appendix B. For information about the authentication modes, see "How To Arrange Network Security" on page 99.

To Troubleshoot NetLDI Startup Failures

If the NetLDI service fails to start in response to a **startnetldi** command, it's likely that the cause is one of the following:

- The NetLDI is to run as root but the guest mode option is specified. This combination is not allowed.
- The account starting the NetLDI does not have permission to create or append to its log file.

- The account starting the NetLDI does not have read and execute permission for `$GEMSTONE/sys/netldid`.

Inspect the NetLDI log for clues. By default, the NetLDI log (*netLdiName*`.log`) is located in `/opt/gemstone/log`. On some systems, this file may be located in `/usr/gemstone/log`, and may be overridden explicitly to `$GEMSTONE_GLOBAL_DIR/log`.

4.3 To List Running Servers

The **gslist** utility lists all Stone repository monitors, shared page cache monitors, and NetLDIs that are running. The **gslist** command by itself checks the locks directory (`/opt/gemstone/locks`, `/usr/gemstone/locks`, or `$GEMSTONE_GLOBAL_DIR/locks`) for entries. The **-v** option causes it to verify that each process is alive and responding. For example:

```
% gslist -v
Status Version Owner     Started      Type   Name
------ ------- --------- ------------ ------ ----
 OK   3.0.0    gsadmin   Mar 26 12:02 cache  gs64stone~1c9fa07f0412665
 OK   3.0.0    gsadmin   Mar 26 12:02 Stone  gs64stone
 OK   3.0.0    gsadmin   Mar 26 10:13 Netldi gs64ldi
```

By default, **gslist** lists servers on the local node. The **-m** *host* option performs the operation on node *host*, which must have a compatible NetLDI running.

4.4 How To Start a GemStone Session

This section tells how to start a GemStone session and log in to the repository monitor. The instructions apply to all logins from the node on which the Stone repository monitor is running.

- For additional information about the GemStone administrative logins, see Chapter 6, "User Accounts and Security."
- For additional information about logging in from a remote node, see Chapter 3, "Connecting Distributed Systems."

This section begins with a brief discussion of environmental variables, and then presents two examples. The first example starts a linked application and logs in to GemStone. The second example starts an RPC application, which in turn spawns a separate Gem session process that communicates with the GemStone server.

The examples use Topaz as the application because it is part of the standard GemStone Object Server distribution. Other applications may use different steps to accomplish the same purpose. Some users may prefer to make these steps part of an initialization file.

For an explanation of the difference between linked and RPC sessions, see "Linked and RPC Applications" on page 78.

To Define a GemStone Session Environment

In order to start a GemStone session, the following must be defined through your operating system environment:

- Where GemStone is installed

  All GemStone users must have a GEMSTONE environment variable that points to the GemStone installation directory, such as */installDir*/`GemStone64Bit3.0.0-x86_64.Linux`. The directory `$GEMSTONE/bin` should be in your search path for commands. For an example, see the next topic, "To Start a Linked Session".

- Which configuration parameters to use

  Because each GemStone session can have its own configuration file, some users may need a second environmental variable, such as GEMSTONE_EXE_CONF. If no other file is found, the session uses system defaults. For further information, see "How GemStone Uses Configuration Files" on page 350.

To Start a Linked Session

The following steps show how to start a linked application (here, the linked version of Topaz). The steps for setting the GEMSTONE environment variable and the operating system path for a session are the same as those given on page 128 for starting a repository monitor. They are repeated here for convenience.

The procedure assumes that the Stone repository monitor has already been started and has the default name *gs64stone*.

Step 1. Set the GEMSTONE environment variable to the *full pathname* (starting with a slash) of the directory where GemStone is installed. Ordinarily this directory has a name like `GemStone64Bit3.0.0-sparc.Solaris`. For example:

```
$ GEMSTONE=/installDir/GemStone64Bit3.0.0-sparc.Solaris
$ export GEMSTONE
```

If you have been using another version on GemStone, be sure you update or delete previous settings of these environment variables:

- GEMSTONE
- GEMSTONE_SYS_CONF
- GEMSTONE_EXE_CONF
- GEMSTONE_NRS_ALL

Step 2. Set your UNIX path. One way to do this is to use one of the `gemsetup` scripts. There is one version for users of the Bourne and Korn shells and another for users of the C shell. These scripts also set your man page path to include the GemStone man pages.

(Bourne or Korn shell)

```
$ . $GEMSTONE/bin/gemsetup.sh
```

or (C shell)

```
% source $GEMSTONE/bin/gemsetup.csh
```

Step 3. Start linked Topaz:

```
% topaz -l
```

Step 4. Set the *UserName* login parameter:

```
topaz> set username DataCurator
```

Step 5. Log in to the Gem session.

```
topaz> login
GemStone Password?
[Info]: LNK client/gem GCI levels = 842/842
[Info]: User ID: DataCurator
[Info]: Repository: gs64stone
[Info]: Session ID: 5
[Info]: GCI Client Host: <Linked>
[Info]: Page server PID: -1
[Info]: Login Time: 03/26/11 12:18:29.634 PDT
[03/26/11 12:18:29.650 PDT]
  gci login: currSession 1 rpc gem processId -1 OOB keep-
alive interval -1
successful login
topaz 1>
```

At this point, you are logged in to a Gem session process, which is linked with the application. The session process acts as a server to Topaz and as a client to the Stone. Information about Topaz is in the manual *GemStone Topaz Programming Environment*.

When you are ready to end the GemStone session, you can log out of GemStone and exit Topaz in one step by invoking the Topaz **exit** command:

```
topaz 1> exit
```

To Start an RPC Session

The following steps show how to start an RPC application (here, the RPC version of Topaz) on the server node. The procedure assumes that the Stone is running under the default name *gs64stone* and that you are already set up to run a GemStone session as described in Steps 1 and 2 of the previous example ("To Start a Linked Session").

Step 1. Use **gslist** to find out if a NetLDI is already running. The default name for the NetLDI is `gs64ldi`. (This list also shows the Stone and shared page cache monitor.)

```
% gslist
Status Version  Owner      Started    Type  Name
------ -------- --------- ------------ ------ ----
exists 3.0.0    gsadmin   Mar 26 12:02 cache gs64stone~1c9fa07f0412665
exists 3.0.0    gsadmin   Mar 26 12:02 Stone  gs64stone
exists 3.0.0    gsadmin   Mar 26 10:13 Netldi gs64ldi
```

Step 2. If necessary, start a NetLDI:

- To start the NetLDI for password authentication, make sure that `$GEMSTONE/sys/netldid` is owned by root and has the S bit set. Issue this command (on some operating systems, you may have to issue it as root):

  `% startnetldi`

- To start the NetLDI in guest mode (authentication is not required), make sure that `$GEMSTONE/sys/netldid` does NOT have the S bit set. Log in as the captive account *name*, then issue this command:

  `% startnetldi -g -a`*name*

Step 3. Unless the NetLDI is running in guest mode with a captive account, decide how you will provide authentication so that the NetLDI can start the Gem session process. Choose one of the following:

- You can create a `.netrc` file in the your home directory containing a line like the following, where *hostName* is the name of this node (which is also the server node):

  `machine` *hostName* `login` *yourUnixId* `password` *yourPassword*

- You can set the application login parameters, such as HostUserName and HostPassword, after you start the application. For example:

  ```
  topaz> set hostusername yourUnixId
  topaz> set hostpassword yourPassword
  ```

Step 4. Start the RPC application (such as Topaz), then set the UserName.

```
topaz> set username DataCurator
```

Step 5. Set GemNetId (the name of the Gem service to be started) to `gemnetobject`. This script starts the separate Gem session process for you. For example:

```
topaz> set gemnetid gemnetobject
```

Step 6. Log in to the GemStone session.

```
topaz> login
GemStone Password?
[03/26/11 12:21:21.267 PDT]
  gci login: currSession 1 rpc gem processId 28846 OOB
keep-alive interval 0
successful login
topaz 1>
```

At this point, you are logged in through a separate Gem session process that acts as a server to Topaz RPC and as a client to the Stone repository monitor.

When you are ready to end the GemStone session, you can log out of GemStone and exit Topaz by in one step by invoking the Topaz **exit** command:

```
topaz 1> exit
```

To Troubleshoot Session Login Failures

Several factors may prevent successful login to the repository:

- Your GemStone key file may establish a maximum number of user sessions that can simultaneously be logged in to GemStone. (Note that a single user may have multiple GemStone sessions running simultaneously.) The limit itself is encoded in `$GEMSTONE/sys/gemstone.key`, but you can examine the comment in that file. For example:

  ```
  # Stone Session limit:      10
  ```

- The GemStone configuration option STN_MAX_SESSIONS (page 385) can restrict the number of logins to fewer than a particular key file allows. An entry in the Stone log file shows the maximum at the time the Stone started. By default, the Stone log file is `$GEMSTONE`/data/*gemStoneName*`.log`. Look for a line like this in a box:

```
SESSION CONFIGURATION: The maximum number of concurrent sessions is 41
```

- The GemStone configuration option SHR_PAGE_CACHE_NUM_PROCS (page 373) restricts the number of sessions that can attach to a particular shared page cache. This number can be different on each node, depending on the configuration file that is read by the process that starts the cache. On the node where the Stone runs, one of this number is used by the Stone, the shared page cache monitor, each GcGem (garbage collection) session, each Stone AIO page server, the page manager, the SymbolGem, and each free frame page server. On other nodes, the Stone's page server and the shared page cache monitor each use one. For details, see "To Set the Page Cache Options and the Number of Sessions" on page 38. Check the Stone's log for warnings that the value requested for SHR_PAGE_CACHE_NUM_PROCS has been adjusted to match your system's configuration.
- The UNIX kernel must provide two semaphores for each session that wants to attach to the shared page cache. See "Reviewing Kernel Tunable Parameters" on page 37.
- The UNIX kernel file descriptor limit can restrict the number of sessions, and GemStone executables attempt to raise that limit. For information, see the discussions of "Estimating File Descriptor Needs" on page 36 (for the Stone) and page 81 (for Gems). On some operating systems, you can examine the kernel limit by invoking **limit**.

- The owner of the Gem or a linked application process must have write access to the extent file and to the shared page cache. Use the UNIX command **ipcs -m** to display permissions, owner, and group for shared memory. For example:

```
server% ipcs -m
IPC status from <running system> as of Mon March 21
16:21:08 PDT 2011
T      ID      KEY         MODE         OWNER      GROUP
Shared Memory:
m      25089   0x4c000ed5  --rw-rw----  gsadmin    users
```

  Typical problems occur with linked applications, which may be installed without the S bit and therefore rely on group access to the shared page cache and the repository.

- If the session is using a separate (RPC) gem process — even on the same node — see "Troubleshooting Remote Logins" on page 121.

4.5 How To Identify Sessions Logged In

Privileges required: SessionAccess.

To identify the sessions currently logged in to GemStone, send the message `System class>>currentSessionNames`. This message returns an array of internal session numbers and the corresponding UserId. For example:

```
topaz 1> printit
System currentSessionNames
%
session number: 2    UserId: GcUser
session number: 3    UserId: GcUser
session number: 4    UserId: SymbolUser
session number: 5    UserId: DataCurator
```

The session number can be used with other System class methods to stop a particular session or to obtain its UserProfile. See `stopSession:`*aSessionId*, `terminateSession:`*aSessionId* `timeout:`*seconds* and `userProfileForSession:`*aSessionId*.

NOTE

Be aware that it may take as long as a minute for a session to terminate after you send `stopSession:`*.If the Gem is responsive, it usually terminates within milliseconds. However, if a Gem is not active (for*

example, sleeping or waiting on I/O), the Stone waits one minute for it to respond before forcibly logging it out. You can bypass this timeout by sending `terminateSession:timeout:`

The method `System class>>descriptionOfSession:`*aSessionId* returns an array of descriptive information by which you can trace the session name to a particular person: the second element shows the operating system process id (pid), and the third element shows the name of the node on which it is running. In this example, the DataCurator session is running on "node1" as pid 3010:

```
topaz 1> printit
System descriptionOfSession: 2
%
an Array
  #1 an UserProfile
  #2 3010
  #3 node1
  ...
```

For details about these methods and the information returned, see the class and method comments in the image.

4.6 How To Shut Down the Object Server and NetLDI

Privileges required: SystemAccess and SystemControl.

To shut down GemStone from UNIX, first make sure that all user sessions have logged out. One way to find out about other user sessions is to send the message `currentSessionNames` to System. For example, using Topaz:

```
topaz 1> printit
System currentSessionNames
%
session number: 2    UserId: GcUser
session number: 3    UserId: GcUser
session number: 4    UserId: SymbolUser
session number: 5    UserId: DataCurator
```

After all user sessions have logged out, use the **stopstone** command, which performs an orderly shutdown in which all committed transactions are written to the extent files.

```
% stopstone [gemStoneName] [-i]
```

If you do not supply the name of the Stone repository monitor, **stopstone** prompts you for one. The default name during startup was `gs64stone`. If necessary, use **gslist** (page 403) to find the name.

The **-i** option aborts all current (uncommitted) transactions and terminates all active user sessions. If you do not specify this option and other sessions are logged in, GemStone will not shut down and you will receive a message to that effect.

stopstone prompts you to supply a GemStone username and password. The user must have the SystemControl privilege (initially, this privilege is granted to SystemUser and DataCurator). For details about user accounts and privileges, see Chapter 6.

There is a similar command to shut down the NetLDI network service.

`% stopnetldi [netLdiName]`

For more information about the stopstone and stopnetldi commands, refer to Appendix B, "GemStone Utility Commands."

If you are logged in to a GemStone session, you can invoke `System class>>shutDown`, which also requires the SystemControl privilege.

CAUTION

*If you must halt a specific Gem session process or GemStone server processes, be sure to use only **kill** or **kill -term** so that the Gem can perform an orderly shutdown.*

*Do NOT use **kill -9** or another uncatchable signal, which may not result in a clean shutdown or may cause the Stone repository monitor to shut down when you intended to kill only a Gem process. If for some reason you need to send **kill -9** to a shared page cache monitor, use **ipcs** and **ipcrm** to identify and free the shared memory and semaphore resources for that cache. If you send **kill -9** to a Stone, use **ipcs** to determine whether **ipcrm** should be invoked.*

4.7 How To Recover from an Unexpected Shutdown

GemStone is designed to shut down in response to certain error conditions as a way of minimizing damage to the repository. If GemStone stops unexpectedly, it probably means that one of the following situations has occurred:

- Disk failure
- Shared page cache monitor failure
- Fatal error detected by a Gem
- File system corruption
- Power failure
- Operating system crash

When GemStone shuts down unexpectedly, check the message at the end of the Stone log file to begin diagnosing the problem. Unless you have set up an environment variable, or specified another file on the **startstone** command line, the Stone log is `$GEMSTONE/data/`*gemStoneName*`.log`.

The `$GEMSTONE/data` directory also contains log files for the Stone child processes. The child processes have log names formed from *gemStoneName,* the process id, and a descriptive abbreviation. For instance:Once the problem is

`gs64stone.log`	Stone repository monitor
`gs64stone_14033admingcgem.log`	Admin GcGem
`gs64stone_2963pcmon.log`	Shared page cache monitor
`gs64stone_2967pgsvrff.log`	Free Frame page server
`gs64stone_2984pgsvraio.log`	AIO page server
`gs64stone_2987pagemanager.log`	Page Manager
`gs64stone_2992reclaimgcgem0-6.log`	Reclaim GcGem
`gs64stone_2994symbolgem.log`	SymbolGem

identified, your recovery strategy should take into account the interdependence of GemStone system components. For instance, if an extent becomes unavailable, to restart the system and recover you may have to kill the Stone repository monitor if it is still running. The **stopstone** command won't work in this situation, since the orderly shutdown process requires the Stone to clean up the repository before it stops.

Normal Shutdown Message

If you see a shutdown message in the system log file, GemStone has stopped in response to a **stopstone** command or a Smalltalk `System shutdown` method:

```
--- 03/21/11 13:00:43 PDT ---
   LoginsSuspended is set to 1 by DataCurator from session 5

   SHUTDOWN command was received from user DataCurator
   session 5 gem processId 29188.
   Waiting for aiowrites to complete
   Waiting for NetRead thread to stop

   Now stopping GemStone.
```

After a normal shutdown, restart GemStone in the usual manner. For instructions, see "How To Start the GemStone Server" on page 128 of this chapter.

Disk Failure or File System Corruption

GemStone prints several different disk read error messages to the GemStone log file. For example:

```
   Repository Read failure,
   fileName = !#dbf!/users/gemstone/data/extent0.dbf
   PageId = 94
   File = /users/gs64stone/data/extent0.dbf
   too few bytes returned from read()
   DBF Operation Read; DBF record 94, UNIX codes: errno=
   34,...
      "A read error occurred when accessing the repository."
```

If you see a message similar to the above, or if your system administrator identifies a disk failure or a corrupted file system, try to copy your extents to another node or back them up immediately. The copies may be bad, but it is worth doing, just in case. If you're lucky, you may be able to copy them back after the underlying problem is solved and start again with the current committed state of your repository.

Otherwise, the procedure you need to follow depends on what was done at the operating system level. For a discussion of the options, see the section "How To Recover After Repair of the File System" on page 291.

Shared Page Cache Error

If you find a message similar to the following in the GemStone log, the shared page cache (SPC) monitor process (`shrpcmonitor`) died. The SPC monitor log, `$GEMSTONE/data/`*gemStoneName*`_pcmon`*nnnn*`.log`, may indicate the reason.

```
--- 03/24/11 15:07:19 PDT ---
    The stone's connection to the local shared cache monitor
was lost.
    Error Text: 'Network partner has disconnected.'
```

The unexpected shutdown of a Gem process may, in rare cases, result in a "stuck spin lock" error that brings down the shared page cache monitor and the Stone. GemStone uses spin locks to coordinate access to critical structures within the cache. In most cases, the monitor can recover if a Gem dies while holding a spin lock, but not all spin locks can be recovered safely. Stuck spin locks may result from a Gem crash, but a typical cause is the use of **kill -9** to kill an unwanted Gem process. If you must halt a Gem process, be sure to use only **kill** or **kill -TERM** so that the Gem can perform an orderly shutdown.

Use **`startstone`** to restart GemStone. For instructions, see "How To Start the GemStone Server" on page 128.

Fatal Error Detected by a Gem

If a Gem session process detects a fatal error that would cause it to halt and dump a core image, the Stone repository monitor may do the same when it is notified of the event. This response on the part of the Stone is configurable through the STN_HALT_ON_FATAL_ERR option (page 381). When that option is set to True and a Gem encounters a fatal error, the Stone prints a message like this in its log file:

```
Fatal Internal Error condition in Gem
   when halt on fatal error was specified in the config file
```

By default, STN_HALT_ON_FATAL_ERR is set to False. That setting causes the Stone to attempt to keep running if a Gem encounters a fatal error; it is the recommended setting for GemStone in a production system. You can set STN_HALT_ON_FATAL_ERR to True during development and testing to provide additional checks for potential risks.

Some Other Shutdown Message

In the event of other shutdown messages in the GemStone log:

1. Consider whether the shutdown might have been caused by a disk failure or a corrupt file system, especially if you see an unexpected message such as `Object not found`. If you suspect one of these conditions, start with a page audit of the repository file (see "How To Audit the Repository" on page 165).

   If the page audit fails, refer to "Disk Failure or File System Corruption" on page 147 of this chapter, and consult your operating system administrator.

   If the audit succeeds, continue to the next step.

2. If you don't suspect disk failure or a corrupt file system, try using **startstone** to restart GemStone. For instructions, see "How To Start the GemStone Server" on page 128.

3. If the restart fails, you may have to restore the repository. For details, see the restore procedures in Chapter 9.

No Shutdown Message

If the GemStone log doesn't contain a shutdown message, there has probably been a power failure or an operating system crash. In that event, the Stone repository monitor automatically recovers committed transactions the next time it starts. Use **startstone** to restart GemStone, as described on page 128. For information about the **startstone** command, see page 411.

4.8 How To Bulk-Load Objects

During bulk loading of objects into the repository, it may be desirable to make the following changes:

- You may need to commit incrementally, but if so, commit as seldom as possible without running out of memory. There is a limit on how large a transaction can be, either in terms of the total size of previously committed objects that are modified, or of the total size of temporary objects that are transitively reachable from modified committed objects.

  To address this concern, increase the GEM_TEMPOBJ_CACHE_SIZE configuration option. The size of each transaction (the number of 16 KB pages written) should be approximately 1/3 to 1/2 the size of

GEM_TEMPOBJ_CACHE_SIZE, and no more than 1/4 to 1/2 the size of the shared page cache.

- If you are loading through GemBuilder, you can reduce growth of your Smalltalk image by using forwarders or explicit stubbing. For instance, when adding objects to a large collection, make the Collection object a forwarder or, after adding each element, send it the message `#stubYourself`.
- Disable epoch garbage collection, if it was enabled (it is disabled by default). This step saves the CPU time ordinarily devoted to scanning for dereferenced objects. To do this, log in as GcUser and set `#epochGcEnabled` to False.
- Alternatively, you can increase performance during bulk loads by adding the following entries to your configuration file:

  ```
  STN_TRAN_LOG_DIRECTORIES = /dev/null, /dev/null;
  STN_TRAN_FULL_LOGGING = TRUE;
  ```

  For information about these configuration file options, see pages 391 and 390, respectively.

  NOTE

  Be aware that using `/dev/null` *for the tranlog directories will prevent you from being able to restore tranlogs in the event of a system failure.*

4.9 Considerations for Large Repositories

GemStone/S 64 allows you to define a very large shared page cache, thereby enabling you to run very large repositories. This section presents special considerations that apply to large repositories.

Loading the object table at startup

When starting the repository, the object table is not loaded into memory, and initial accesses can take an excessively long time. If you encounter degraded application performance for a period after restarting the Stone, you may want to start up using cache warming. There is an initial heavy I/O load at Stone startup with cache warming, but subsequent application performance should be consistent with your normal application performance. You may choose to preload just the object table pages, which are most important for performance. Alternatively, you can also preload data pages, which will improve performance if you have a large cache with a relatively fixed working set of data pages.

There are two ways to perform cache warming on startup.

- The configuration parameters STN_CACHE_WARMER and STN_CACHE_WARMER_SESSIONS enable and configure cache warming, respectively. See page 375 for more information.
- Alternatively, you may run the **startcachewarmer** utility. For details about **startcachewarmer**, see page 407.

Making efficient use of remote caches

When running a system on which many users log in simultaneously, consider using remote caches so that you don't need to run all Gem processes on the same machine. There are a couple of ways to optimize this. The following configuration options are of particular interest:

- To allow Gems to make more efficient use of the large cache, set the GEM_PGSVR_FREE_FRAME_CACHE_SIZE configuration option (page 366) to increase the size of the Gem free frame cache. For example:

  ```
  GEM_PGSVR_FREE_FRAME_CACHE_SIZE = 25;
  ```

- To improve performance on remote caches, set GEM_PGSVR_UPDATE_CACHE_ON_READ (page 367) to True so that remote Gem sessions will update their local caches. For example:

  ```
  GEM_PGSVR_UPDATE_CACHE_ON_READ = TRUE;
  ```

Disk Space and Commit Record Backlogs

Sessions only update their view of the repository when they commit or abort. The repository must keep a copy of each session's view so long as the session is using it, even if other sessions frequently commit changes and create new views (commit records). Storing the original view and all the intermediate views uses up space in the repository, and can result in the repository running out of space. To avoid this problem, all sessions in a busy system should commit or abort regularly.

For a session that is not in a transaction, if the number of commit records exceeds the value of STN_CR_BACKLOG_THRESHOLD, the Stone repository monitor signals the session to abort by signaling `TransactionBacklog` (also called "sigAbort"). If the session does not abort, the Stone repository monitor reinitializes the session or terminates it, depending on the value of STN_GEM_LOSTOT_TIMEOUT.

Sessions that are in transaction are not subject to losing their view forcibly. Sessions in transaction enable receipt of the signal `TransactionBacklog`, and handle it appropriately, but it is optional. It is important that sessions do not stay in transaction for long periods in busy systems; this can result in the Stone running out of space and shutting down. However, sessions that run in automatic

transaction mode are *always* in transaction; as soon as they commit or abort, they begin a new transaction. (For a discussion of automatic and manual transaction modes, see the "Transactions and Concurrency Control" chapter of the *GemStone/S 64 Bit Programming Guide*.)

To avoid running out of disk space, we recommend that you use *manual transaction mode* whenever possible. To enter manual transaction mode:

```
topaz> printit
System transactionMode: #manualBegin
%
```

At the point that this session needs to commit a change, begin a transaction manually, then make the changes:

```
topaz> printit
System beginTransaction .
AllUsers addNewUserWithId: #Jane password: 'gemstone' .
System commitTransaction
%
```

After you commit (or abort) the transaction, your session will return to waiting outside of a transaction.

Handling signals indicating a commit record backlog

Even in manual transaction mode, it is possible to cause a commit record backlog, depending on how your system is configured. Sessions should ensure that they commit or abort regularly, or set up sigAbort handlers to abort when requested by the Stone. A sigAbort handler may be as simple as this:

Example 4.1 sigAbort handler

```
Exception
   installStaticException:
   [ :exception :GSdictionary :errID :array |
      System abortTransaction.
      System enableSignaledAbortError).
```

Note that a session that is entirely idle does not become aware of the signal to abort, and may timeout and be terminated by the stone in spite of the handler. If your application may have idle sessions, we recommend setting up a timer that causes regular aborts when the session is otherwise idle.

Sessions that are in transaction, and therefore immune from the sigAbort mechanism, may also be signaled when there is a commit record backlog. When the number of commit records exceeds the value of STN_CR_BACKLOG_THRESHOLD, and the session holding the oldest commit record is in transaction, the Stone repository monitor signals the session by sending `TransactionBacklog`. The session then has the opportunity to perform a continueTransaction to update its view of unmodified objects. It may also commit or abort. Unlike sigAbort, the session can choose to ignore this message and will not receive further signals from the stone.

For more information on these signals, see the *Programming Guide* for GemStone/S 64 Bit.

Chapter

5 Monitoring GemStone

This chapter tells you:

- Where to look for the log files that GemStone/S 64 Bit processes create
- How to audit the repository
- How to analyze the repository contents
- How to monitor the performance of the GemStone server and its clients using GemStone Smalltalk methods

If you decide to keep a GemStone session running for occasional use, be careful not to leave it in an active transaction. A prolonged transaction can cause an excessive commit record backlog and impede garbage collection activity, resulting in undesirable repository growth, until you either commit or abort.

NOTE

For sessions that are not committing changes to the repository, we recommend that monitoring be done in manual transaction mode or transactionless mode. For details on entering and using manual transaction mode, see page 151.

5.1 GemStone System Logs

In addition to transaction logs, GemStone creates three types of log files:

- Logs for GemStone object server processes (page 156)
- Logs for processes related to individual GemStone sessions (page 161)
- Logs for GemStone network server processes, NetLDIs (page 163)

If a GemStone server is running, you can use the **gslist** utility to locate its logs. Use **gslist -x** to display the location of the current log file for Stones, NetLDIs, and the shared page cache monitors.

The logs for the AIO page servers, free frame page servers, SymbolGem, Page Manager, and Admin and Reclaim GcGems are in the same location as the corresponding Stone's log.

WARNING

The Stone writes several files to the `/opt/gemstone/locks` *or equivalent directory (as discussed on page 38). These log files are usually regenerated if deleted, but we recommend not removing them manually.Use* ***gslist -c*** *to clear out unnecessary lock files.*

For more about the **gslist** command, see page 403.

GemStone Server Logs

The Stone repository monitor and its child processes each create a log file in a single location. By default, the files are in `$GEMSTONE/data` and have a name beginning with the Stone name. Table 5.1 shows typical log names for a Stone with the default name of `gs64stone`. Log names for child processes also include the process id and a descriptive suffix.

Table 5.1 Representative Log Names for GemStone Server Processes

`gs64stone.log`	Stone repository monitor
`gs64stone_14033admingcgem.log`	Admin GcGem
`gs64stone_2963pcmon.log`	Shared page cache monitor
`gs64stone_2967pgsvrff.log`	Free frame page server
`gs64stone_2984pgsvraio.log`	AIO page server
`gs64stone_2987pagemanager.log`	Page Manager
`gs64stone_2992reclaimgcgem0-6.log`	Reclaim GcGem
`gs64stone_2994symbolgem.log`	Symbol Gem

Several factors can alter the name and location of these logs. The precedence is

1. A path and filename supplied by **startstone -l** *logFile. logfile* may be a filename, or a relative or absolute path and filename, to which the account starting the Stone has write permission. If *logFile* is a filename without a path, *logFile* is created in the current directory. Logs for the child processes in Table 5.1 are placed in the same directory.
2. A path and filename specified by the GEMSTONE_LOG environment variable. As with **startstone -l**, this may be set to a filename or to a relative or absolute path and filename. Child process log files are created in the same location.
3. `$GEMSTONE/data/`*gemStoneName*`.log`.

Log file deletion on shutdown

When a GemStone server shuts down, some server process log files are deleted. Other files are retained in case they are needed later for tuning or problem diagnosis. Table 5.2 details the specific behavior for each process's logs, and how to change that behavior in cases where the behavior is configurable.

Log files are never deleted if the process exits abnormally. In such cases, the log file may be requested when you contact GemStone Technical Support.

Since some logs are not deleted, if you restart GemStone often, you may need to manually remove log files periodically.

Table 5.2 Log file handling by process type

Stone	The same log file is appended to on restart. Log file is never deleted.

Table 5.2 Log file handling by process type (Continued)

Shared Page Cache Monitor	A new log file is created on restart, including the process PID. Log file is never deleted on exit.
Page Manager	A new log file is created on restart, including the process PID. Log file is not deleted by default on normal exit. To force this log to be deleted on normal exit, edit `$GEMSTONE/sys/runpagemgrgem` to look like this: `unset $GEMSTONE_CHILD_LOG.`
Symbol Gem	A new log file is created on restart, including the process PID. Log file is not deleted by default on normal exit. To force this log to be deleted on normal exit, edit `$GEMSTONE/sys/runsymbolgem` to look like this: `unset $GEMSTONE_CHILD_LOG.`
Free frame page servers, AIO page servers	A new log file is created on restart, including the process PID. Log file is deleted by default on normal exit. To prevent this log from being deleted on normal exit, edit `$GEMSTONE/sys/runpgsvrmain` to look like this: `# unset $GEMSTONE_CHILD_LOG.` This will affect all page servers, free frame, AIO page servers, and page servers for remote gems.
Admin GcGem	A new log file is created on restart, including the process PID. Log file is deleted by default on normal exit. To prevent this log from being deleted on normal exit, edit `$GEMSTONE/sys/runadmingcgem` to look like this: `# unset $GEMSTONE_CHILD_LOG.`
Reclaim GcGem	A new log file is created on restart, including the process PID. Log file is deleted by default on normal exit. To prevent this log from being deleted on normal exit, edit `$GEMSTONE/sys/runreclaimgcgem` to look like this: `# unset $GEMSTONE_CHILD_LOG.`

Stone Log

The log for the Stone repository monitor is always appended to, and is therefore cumulative across runs by default. This log is the first one you should check when a GemStone system problem is suspected. In addition to possible warnings and error messages, the log records the following useful information:

- The GemStone version.
- The configuration files that were read at startup and, if DUMP_OPTIONS is set to True (page 362), the resulting Stone configuration.
- Each startup and shutdown of the Stone, the reason for the shutdown, and whether recovery from transaction logs was necessary at startup.
- Each expansion of a repository extent and its current size.
- Each opening of a new transaction log.
- Each startup and shutdown of the GcGem(s), and the corresponding processId.
- Each #abortErrLostOtRoot sent to a Gem.
- Each suspension and resumption of logins.
- Certain changes to the login security system.
- Each time a backup is started and when the backup is completed.

Admin GcGem Logs

Each time the Stone repository monitor starts an administrative garbage collection session (Admin GcGem) process, a new log is created in the same location as the Stone's log. The log name is formed using the pattern:

*stoneName*_*PID***admingcgem.log**

where *stoneName* is the name of the Stone, and *PID* is the process Id of the Admin GcGem process.

This log shows the startup value of the Admin GcGem parameters that are stored in GcUser's UserGlobals, and any changes to them, and records other Admin GcGem functions.

Shared Page Cache Monitor Log

The log for the shared page cache monitor on the Stone's machine is located in the same directory as the Stone's log. This log file has a name of the form

*stoneName*_*PID***pcmon.log**

Check this log if other messages refer to a shared page cache failure.

When a session logs in from another node, a log is created for the shared page cache monitor on the remote node. This log is located by default in the home

directory of the account that started the Stone, but this location can be modified by environment variable settings. The default name is of the form

`startshrpcmon`*PIDNode*`.log`

where *PID* is the process Id of the monitor process, and *Node* is the name of the remote node.

Among the items included in the log for the shared page cache monitor are:

- Its configuration (for remote nodes, this may be different from the configuration on the Stone's node).
- The number of processes that can attach (which can limit the number of logins).
- The UNIX identifiers for the memory region and the semaphore array (these identifiers are helpful in the event you must remove them manually using the **ipcrm** command).

Free Frame Page Server Log

One or more free frame page servers started up on repository startup. Each one has an individual log file, located in the same directory as the log for the shared page cache monitor. These log files have names of the form

stoneName`_`*PID*`pgsvrff.log`

where *stoneName* is the name of the stone, and *PID* is the process Id of the free frame page server process.

These logs ordinarily are not of interest unless they contain an error message.

AIO Page Server Log

One or more AIO page servers are started up on repository startup. Each one has an individual log file, located in the same directory as the log for the shared page cache monitor. These log files have names of the form

stoneName`_`*PID*`pgsvraio.log`

where *stoneName* is the name of the stone, and *PID* is the process Id of the AIO page server process.

These logs ordinarily are not of interest unless they contain an error message.

Page Manager Log

The Page Manager log is located in the same directory as the log for the shared page cache monitor. This log file has a name of the form

*stoneName*_*PID*`pagemanager.log`

where *stoneName* is the name of the stone, and *PID* is the process Id of the Page Manager process.

Ordinarily these logs are not of interest unless errors occur or tuning is required.

Reclaim GcGem Logs

Each time the Stone repository monitor starts a new reclaim garbage collection session (Reclaim GcGem) process, a new log is created in the same location as the Stone's log. The log name is formed using the pattern:

*stoneName*_*PID*`reclaimgcgem`*N*`-`*M*`.log`

where *stoneName* is the name of the stone, *PID* is the process Id of the Reclaim GcGem process, and *N* and *M* indicate the range of extents for this Reclaim GcGem. A Reclaim GcGem acts on a range of extent files; these digits represent the start and stop indices (starting with 0) of the range for this Reclaim GcGem. In Table 5.1, the log name (`gs64stone_2992reclaimgcgem0-6.log`) indicates that the Reclaim GcGem is running on the first seven extents of the repository (0–6). If there is only one extent, the "`-`*M*" is omitted.

This log shows the startup value of the Reclaim GcGem parameters that are stored in GcUser's UserGlobals, and any changes to them, and records other Reclaim GcGem functions.

Symbol Gem Log

The Symbol Gem log is located in the same directory as the Stone's log. This log file has a name of the form

*stoneName*_*PID*`symbolgem.log`

where *stoneName* is the name of the stone, and *PID* is the process Id of the Symbol Gem process.

Ordinarily these logs are not of interest unless an error has occurred.

Logs Related to Gem Sessions

Except for linked session that are running on the same node as the Stone, login depends on NetLDI services to spawn one or more supporting processes. In each

case, the NetLDI creates a log file that includes in its name the identity of the node on which the process is running.

Linked logins do not have separate log files. Log file output is sent to stdout of the linked process.

All RPC logins spawn a Gem session process.

For RPC logins where the Gem is not on the same node as the Stone, or for linked logins that are not on the same node as the Stone, the following additional processes are also spawned:

- A page server (for the session) to access a repository extent on the server node.
- A page server (for the Stone) to start or access a shared page cache on the client's node.
- A shared page cache monitor (for the Stone) to manage the cache on the client's node.

By default, the log files for these processes are located in the home directory of the account that owns the corresponding process. For the Gem session process and the page server on the server node, that account ordinarily is the application user. For the shared page cache monitor and page server on the client node, that account is the one that invoked **startstone**.

You can change the default location by setting **#dir** or **#log** in the GEMSTONE_NRS_ALL environment variable for the NetLDI itself or for individual clients (see "To Set a Default NRS" on page 105). Alternatively, when you log in to GemStone, you can specify a different network resource string (NRS) in your login parameters.

Table 5.3 shows typical log names for session-related processes, given a Stone and repository on *node1* with a login from a Gem session process on *node2*.

Table 5.3 Typical Logs Supporting Gem Sessions

Typical Name	GemStone Process
`gemnetobject27853`*node2*`.log`	Gem session process on *node2* (serves an RPC session)
`pgsvrmain27819`*node2*`.log`	Page server on *node2* that the repository monitor uses to create and access its shared page cache on *node2*
`startshrpcmon27820`*node2*`.log`	Shared page cache monitor on *node2*

Table 5.3 Typical Logs Supporting Gem Sessions

Typical Name	GemStone Process
`pgsvrmain12397`*node1*`.log`	Page server on *node1* that the Gem session process uses to access the repository extents on *node1*

If a process shuts down normally, the log file may automatically removed. (See Table 5.2 on page 157 for specific behavior by process type.) Log files are never deleted if the process shuts down abnormally. This way, the log files that remain may provide helpful diagnostic information.

If you want to retain the log even when a Gem session process exits normally, edit the scripts according to the instructions in Table 5.2 on page 157. If the NetLDI on the client node has a separate $GEMSTONE directory, edit the appropriate scripts in the client's installation directory.

NetLDI Logs

Each NetLDI creates a log file (*netLdiName*`.log`) in `/opt/gemstone/log` (or an equivalent; see page 38) on the node on which it runs. This location and name can be overridden by including the option **`-l`***logname* when starting the NetLDI. Each NetLDI you start with the same name appends to one log, so it's a good idea to remove outdated messages occasionally.

By default, the NetLDI log contains only configuration information and error messages. The configuration information reflects the environment at the time the NetLDI was started and the effect of any authentication switches specified as part of the startup command. The following log description for the default configuration may be helpful for comparison:

```
System password authorization is permitted.
Authentication is required only to create processes.
Process creation is permitted through user's HOME directory.
Created processes belong to client's account.
```

The preceding lines map to NetLDI options in this way:

Line 1 - Authentication is not restricted.

Line 2 - Guest mode is not in use (**-g**), but authentication is not required for all NetLDI services (**-s**).

Line 3 - Services are not restricted to those listed in `$GEMSTONE/sys/services.dat` (**-n**).

Line 4 - Captive accounts are not in use (-**a***name*).

In some cases it is helpful to log additional information by starting the NetLDI in debug mode (**startnetldi -d**). The debug log records each exchange between the NetLDI and a client. Because the log becomes much larger, you probably won't want to use this mode routinely.

5.2 How To Audit the Repository

This section describes two levels of checks that you can perform on the repository.

- A *page audit* (page 165) typically is invoked to ensure page-level consistency after some kind of system failure, such as a read-write error or a cache coherency error. In these cases, a successful page audit indicates that the problem did not affect the committed repository. GemStone must be halted when you perform a page audit.
- An *object audit* (page 167) checks the consistency of the repository at the object level. An object audit can be performed as part of routine maintenance and is performed while GemStone is running.

To Perform a Page Audit

Page audits allow you to diagnose problems in the system repository by checking for consistency at the page level.

The **pageaudit** utility can be run only on a repository that is not in use.

Pageaudit scans the rootpages in a repository, pages used in the bitmap structures referenced by the rootpage, and all other pages (including data pages) to confirm page-level consistency. It does not check that the data on data pages is valid. For that, you need to run object audit; see "To Perform an Object Audit and Repair" on page 167.

To check for page-level problems, run **pageaudit** on the repository defined in your ordinary GemStone configuration by issuing this command at the operating system level:

% **`pageaudit`** [*gemStoneName*] [**-e** *exeConfig*] [**-z** *systemConfig*] [**-f**] [**-d**] [**-l** *logfile*] [**-h]**

where:

- *gemStoneName* is the name of the GemStone repository monitor.
- *systemConfig* is the system configuration file (page 350).
- *exeConfig* is the executable configuration file (page 353).
- *logfile* is the name of the output file.

All of these arguments are optional in a standard GemStone configuration. If these options are not supplied, **pageaudit** uses `gs64stone-audit` for *gemStoneName*, and the default system and executable configuration files.

- For more about the **pageaudit** command and its options, see page 405.
- For online documentation, type **`pageaudit -h`** or **`man pageaudit`**.

In addition to the audit results, **pageaudit** prints status updates as it is running, and the output includes repository statistics to the screen. For example:

```
GemStone is starting a page audit of the Repository.
Finished auditing reserved pages in extent   0.
Finished pages past end
Begin auditing current checkpoint.
Finished auditing checkpoint bitmaps.
Finished auditing scavengable pages.
Start auditing object table pages.
Finished auditing object table pages.
Finished auditing commit records.
Finished auditing alloc pages shadowed.
Begin auditing data pages.  Count=970
Finished auditing data pages
PAGE AUDIT STATISTICS paris sun4u (Solaris 2.10
Generic_141444-09) - 01/26/11 17:32:33 PST

16384 bytes = 1 GemStone Page
1048576 bytes = 1 Mbytes
Repository Size                          53 Mbytes
Data Pages                                6 Mbytes
Meta Information Pages                    1 Mbytes
Shadow Pages                              0 Mbytes
Free Space in Repository                 44 Mbytes
**** Number of differences found in page allocation = 0.
   Page Audit of Repository completed successfully.
```

The report contains the following statistics:

Repository Size
: The total physical size of the repository; this is the same size that the operating system reports for an extent file.

Data Pages
: This includes all pages referenced from the object table.

Meta Information Pages
: Pages that contain only internal information about the repository, such as the object table.

Shadow Pages
: Pages scheduled for scavenging by the reclaim task.

Free Space in Repository
: Free space in the repository is computed as the number of free pages times the size of a page (16 KB). That value reflects the number of pages available for allocation to Gem session processes. It excludes space fragments on partially filled data pages.

If the page audit finds problems, the message to the screen ends with a message like this:

```
-------------- PAGE AUDIT RESULTS --------------
**** NumberOfFreePages = 980 does not agree with audit
      results = 988

**** Problems were found in Page Audit.
**** Refer to recovery procedures in System Administrator's
Guide.
```

If there are problems in the page audit, you will need to restore the repository file from backups. (See the section “How To Restore from a Smalltalk Full Backup” on page 269.)

To Perform an Object Audit and Repair

Privileges required: SystemControl.

Object audits check the consistency of the repository at the object level. Starting with Object Table, each object is located and validated.

Object audit is performed using multiple threads (lightweight sessions), and can be configured to perform as quickly as possible using a large amount of system resources, or configured to use fewer resources and take longer to run.

Object audit should be run from linked Topaz, and on the same machine as the Stone.

`Repository >> objectAudit`
`objectAudit` is the normal way to perform the audit. You may have other sessions logged in and running simultaneously, but the audit will impact performance. This audit uses two threads and up to 90% of the CPU.

`Repository >> fastObjectAudit`
`fastObjectAudit` is like `objectAudit`, but is configured to use most or all system resources to complete as quickly as possible. This is useful when running an audit on offline systems.

`Repository >> objectAuditWithMaxThreads:` *maxThreads*
`percentCpuActiveLimit:` *aPercent*
This method allows you to specify the exact performance/impact parameters for the audit, if neither `objectAudit` nor `fastObjectAudit` is satisfactory for your requirements.

If the repository is consistent and no errors are found, the audit will complete with the line:

Object Audit: Audit successfully completed; no errors were detected.

Otherwise, the reasons for failure are reported to standard output. For example:

```
topaz 1> run
SystemRepository repair
%
Object [20897537] references class [27554561] which does not exist
Object [27551745] references class [27554561] which does not exist
In object [27553281] of class Widget [26374657], the logical size 42
is invalid
In object [27554049] of class Widget [26374657], the object format 1
disagrees with the class format 0
Object [27554817] references class [27554561] which does not exist

Object Audit: 5 errors were found
ERROR 3022 , a InternalError occurred (error 3022),
reason:abortErrObjAuditFail (InternalError)
```

Performing the Object Audit

To perform an object audit:

Step 1. Log in to GemStone using linked Topaz (**topaz -l**).

Step 2. Send one of the audit messages to the repository. For example:

```
topaz 1> printit
SystemRepository objectAudit
%
```

Errors During the Object Audit

If errors are reported during the object audit, you may wish to perform a `markForCollection` and `reclaimAll` and repeat the object audit. This may clear up problems if the object (s) that is (are) corrupt are not referenced from any live objects. Whether this is useful will depend on the particular errors reported.

The audit involves a number of checks and specific error messages. The following categories illustrate their nature:

- Object corruption — The object header should contain valid (legal) information about the object's tag size, body size (number of instance variables), and physical size (bytes or OOPs).
- Object reference consistency — No object should contain a reference to a nonexistent object, including references to a nonexistent class.
- Identifier consistency — OOPs within the range in use (that is, up to the high-water mark) should be in either the Object Table or the list of free OOPs, and OOPs for objects existing in data pages should be in the Object Table.

Error Recovery

If the errors are a few invalid object references, you may choose to repair them yourself. Use the Topaz object identity specification format @*identifier* to substitute nil or an appropriate reference for an invalid reference. For example, to replace an invalid reference in an instance of Array:

topaz 1> **send @119873 at: 3 put: nil**

You can have GemStone attempt appropriate repairs during the re-scan by invoking `Repository>>repair`. The following repairs illustrate their nature:

- nil is substituted for an invalid object reference.

- Class String is substituted for an invalid class of a byte object, class Array for a pointer object, or class IdentitySet for a nonsequenceable collection object.
- Oops in the Object Table for which the referenced object does not exist are inserted into the list of free Oops. Oops for which an object exists but which are also in the list of free Oops are removed from the free list.

A descriptive message is displayed for each repair.

To have GemStone make the repairs, do the following:

Step 1. Log in to GemStone using linked Topaz (**topaz -l**).

Step 2. Send the message `repair` to the repository.

```
topaz 1> printit
SystemRepository repair
%
```

The repair audits the repository, keeping track of errors. After the initial audit completes, each error found is repaired. The repair will commit periodically to avoid memory issues, and log off when it is complete. For example:

```
topaz 1> run
SystemRepository repair
%
Object [20897537] references class [27554561] which does not exist
Object [27551745] references class [27554561] which does not exist
In object [27553281] of class Widget [26374657], the logical size 42
is invalid
In object [27554049] of class Widget [26374657], the object format 1
disagrees with the class format 0
Object [27554817] references class [27554561] which does not exist

Object Audit: 5 errors were found
Repairing error: BadClassId - Object [20897537] references class
[27554561] which does not exist
  Changing class to String [74753]
Repairing error: BadClassId - Object [27551745] references class
[27554561] which does not exist
  Changing class to IdentitySet [73985]
Repairing error: BadLogicalSize - In object [27553281] of class
Widget [26374657], the logical size 42 is invalid
  resetting logialSize to 8
```

```
Repairing error: BadFormat - In object [27554049] of class Widget
[26374657], the object format 1 disagrees with the class format 0
  Changing class to String [74753]
Repairing error: BadClassId - Object [27554817] references class
[27554561] which does not exist
  Changing class to Array [66817]

[Info]: Logging out at 04/06/11 10:41:41 PDT
ERROR 4061 , The session is terminating abnormally, completed the
repair of 5 objects, forcing logout.
```

Because `repair` includes an object audit, some administrators prefer to use this method initially rather than repeating the audit if repair is required.

5.3 Profiling Repository Contents

Some questions — such as "what is using up all the space in my Repository?" — can only be answered by examining the types and numbers of objects in your repository. To find out this information, you can use methods on GsObjectInventory.

The methods in GsObjectInventory count all instances of all classes in the repository — or in any collection, or in a hidden set, or in a file of disconnected possible garbage objects — and report the results, ordered by the number of instances or by space consumed.

GsObjectInventory performs a multi-threaded scan of the repository, and thus should only be run in session on the same machine as the Stone. To tune the impact of the scan, additional protocol allows you to perform fast scans or to specify the impact levels. For details, see methods in the image.

The scans require the GcLock, and so cannot be run while any garbage collection operation is running, nor can garbage collection operations be started while a GsObjectInventory scan is going on.

The following code will report the number of instances and the space required for all Classes whose total space requirements are more than 10000 bytes.

```
topaz 1> run
GsObjectInventory profileRepository byteCountReportDownTo: 1000
%
   *** GsObjectInventory byteCountReport printed at: 25/03/2011
14:10:45 ***
Hidden classes are included in this report.
```

```
_____________________________________________________________________
Class                                    Instances          Bytes

_____________________________________________________________________
String                                       22497        8126360
GsNMethod                                    15289        3005728
Array                                        18921        2945904
GsMethodDictionary                            2570        1292696
Symbol                                       15146         658624
LargeObjectNode                                 30         456512
CanonStringBucket                             2016         269984
Class                                         1254         195776
IdentityKeyValueDictionary                    1271         172880
SymbolAssociation                             3923         157416
ExecBlock                                     2312         148128
SymbolDictionary                               766         141008
IdentityCollisionBucket                       1336         131824
SymbolSet                                     3502         103808
GsClassDocumentation                           464          48272
DepListBucket                                  751          42064
DateTime                                       634          35528
ClassHistory                                   625          30176
GsDocText                                      714          28624
LargeInteger                                     9          22976
TimeZoneTransition                             313          17528
CanonSymbolDict                                  1          16176
WordArray                                       13          13920
EqualityCollisionBucket                        128          13248
_____________________________________________________________________
```

The same profiling with an instance count report is much shorter, since the number of instances, rather than the bytes of space used, limits the results.

```
topaz 1> run
GsObjectInventory profileRepository instanceCountReportDownTo: 10000
%
 *** GsObjectInventory instanceCountReport printed at: 25/03/2011
16:17:28 ***
Hidden classes are included in this report.

_____________________________________________________________________
Class                                    Instances          Bytes

_____________________________________________________________________
String                                       22497        8126360
Array                                        18921        2945904
```

```
GsNMethod                                   15289        3005728
Symbol                                      15147         658680
_______________________________________________________________
```

These reports include instances of hidden classes - these are classes that are used to implement internal GemStone objects, which are invisible to the image. One such class is LargeObjectNode. Instances of LargeObjectNodes are used to implement the tree structures that underlie large collections. To avoid seeing hidden classes - which will include the space used by the hidden class within the root, public object, profile using the method `profileRepositoryAndSkipHiddenClasses` rather than `profileRepository`.

For more on GsObjectInventory, see the methods in the image.

5.4 Monitoring Performance

As part of your ongoing responsibilities, you may find it useful to monitor performance of the object server or individual session processes.

GemStone includes graphical tools to allow you to record statistics in file and analyze this data graphically. You can also programmatically access these statistics.

A full list of the statistics that are recorded and are available programmatically can be found in Appendix H, "Cache Statistics", on page 475.

Statmonitor and VSD

GemStone includes the statmonitor utility, which records statistics about GemStone processes to a disk file. You can configure the statistics recorded, how frequently the statistics are collected, and other details.

We recommend running statmonitor at all times, as it provides a valuable record of many aspects of system behavior.

To view this data, VSD (Visual Statistics Display) graphically displays the statistics.

For more details on using statmonitor and VSD, see Appendix G, "statmonitor and VSD Reference", on page 457.

Programmatic Access to Cache Statistics

A set of methods on the System class provide a way for you to analyze performance by programmatically examining the statistics that are collected in the shared page cache. This is the same data that is visible using statmonitor and VSD, although statmonitor and VSD can collect additional OS level information.

A process can only access statistics that are kept in the shared page cache to which it is attached. Sessions that are running on a different node than the Stone use a separate shared cache on that remote node. This means that processes that are on a different node than the Stone, cannot access statistics for the Stone or for other server processes that are attached to the Stone's shared page cache.

Within the shared page cache, GemStone statistics are stored as an array of *process slots*, each of which corresponds to a specific process. Process slot 0 is the shared page cache monitor. On the Stone's shared page cache, process slot 1 is the Stone; on remote caches, slot 1 is the page server for the Stone that started the cache. Subsequent process slots are the page servers, Page Manager, Admin and Reclaim GcGems, Symbol Gem, and user Gems. The order of these slots depends on the order in which the processes are started up, and is different on remote caches.

The specific set of statistics is different for each type of process that can attach to the shared page cache. The types of processes are numbered:

1 = Shared page cache monitor
2 = Stone
4 = Page server
8 = Gem (including Topaz, GBS, and other GCI applications).

Statistics by name

To obtain the value for a specific statistics for the Stone, the Stone's SPC monitor, or for the current session, use the following methods:

```
System class >> stoneCacheStatisticWithName:
System class >>
   primaryCacheMonitorCacheStatisticWithName:
System class >> myCacheStatisticWithName:
```

These methods will return the statistics value corresponding to the given name for that process. If the statistics name is not found, it returns nil.

For example, to retrieve the statistics named 'CommitRecordCount' for the Stone:

```
topaz 1> printit
System stoneCacheStatisticWithName: 'CommitRecordCount'.
```

```
%
23
```

To retrieve the current session's PageReads:

```
topaz 1> printit
System myCacheStatisticWithName: 'PageReads'.
%
548
```

All statistics for a process

The general way to retrieve statistics is as an array of values. To understand what the value at each index refers to, there are corresponding description methods to return an array of Strings. Matching the index of the statistic name to the index within the values locates the value for that statistic.

Since the statistics are different for the different types of processes, you will need to use corresponding methods to collect the statistics and the descriptions.

For the Stone, the Gem that is running the code, and the Stone's shared page cache monitor, no further information is needed to identify them within the cache, so the following pairs of methods can be used:

```
System cacheStatisticsDescriptionForGem.
System myCacheStatistics.

System cacheStatisticsDescriptionForStone.
System stoneCacheStatistics.

System cacheStatisticsDescriptionForMonitor.
System sharedPageCacheMonitorCacheStatistics.
```

For example, while you would normally use `stoneCacheStatisticForName:`, here is another possible way to get the CommitRecordCount:

```
topaz 1> printit
| index |
index := System cacheStatisticsDescriptionForStone indexOf:
'CommitRecordCount'.
System stoneCacheStatistics at: index.
%

23
```

To collect statistics for other Gems, and for page servers, you need to determine the process Id, session Id, or slot of the specific Gem or page server, or the cache name

of the Gem. There are a variety of ways you might determine this, but one way is to examine the results of:

```
System cacheStatisticsForAllSlotsShort
```

This method returns the name, process Id, session Id, statistics type, and process slot for each process currently attached to the cache. For example:

```
topaz 1> printit
(System cacheStatisticsForAllSlotsShort) collect:
   [:ea | ea printString]
%
an Array
  #1 anArray( 'ShrPcMonitor', 7722, 4294967295, 1, 0)
  #2 anArray( 'gs64stone', 7721, 0, 2, 1)
  #3 anArray( 'FreeFrmPgsvr2', 7725, 4294967294, 4, 2)
  #4 anArray( 'AioPgsvr3', 7726, 4294967294, 4, 3)
  #5 anArray( 'PageManager4', 7729, 1, 8, 4)
  #6 anArray( 'GcAdmin5', 7734, 2, 8, 5)
  #7 anArray( 'SymbolGem6', 7735, 3, 8, 6)
  #8 anArray( 'GcReclaim0-1 7', 7733, 4, 8, 7)
  #9 anArray( 'Gem26', 2271, 5, 8, 8)
  #10 anArray( 'Gem27', 16924, 6, 8, 9)
```

Of course, a Gem may log out between the time you execute this and the time you collect statistics, so be sure that your code handles that condition gracefully.

The methods you use to get the statistics and the corresponding descriptions will depend on how you have determined the specific process you want information about.

By name:

System cacheStatisticsForProcessWithCacheName: *aString*
(You must manually determine the process type)

or

System cacheStatsForGemWithName: *aString.*
System cacheStatisticsDescriptionForGem.

By operating system Process Id (PID):

System cacheStatisticsProcessId: *aPid.*
System cacheStatisticsDescriptionAt:
(System cacheSlotForProcessId: *aPid)*.

By process slot:

```
System class >> cacheStatisticsAt: aProcessSlot
System class >> cacheStatisticsDescriptionAt: aProcessSlot
```

By session Id:

The page server for a Gem assumes the same sessionId as its Gem.

```
System gemCacheStatisticsForSessionId: aSessionId.
System cacheStatisticsDescriptionForGem.
```

or

```
System cacheStatsForPageServerWithSessionId: aSessionId
System cacheStatisticsDescriptionForPageServer
```

For example, to find an aggregate value for TimeInFramesFromFindFree of all Gems in the system:

```
topaz 1> printit
| gemPids index time |
gemPids := Array new.
System cacheStatisticsForAllSlotsShort do:
   [:anArray |
   (anArray at: 4) = 8 ifTrue:
         [gemPids add: (anArray at: 2)].
   ].
index := System cacheStatisticsDescriptionForGem indexOf:
   'TimeInFramesFromFindFree'.
time := 0.
gemPids do: [:aPid | | stats |
   stats := System cacheStatisticsProcessId: aPid.
   stats ifNotNil: [time := time + (stats at: index)].
   ].
time
%
0
```

Setting the name for the Gem in the cache

To make it easier for you to track cache statistics for specific Gems, you can explicitly give each Gem a unique name. The method

```
System cacheName: aString
```

sets the name for the current Gem session in the cache statistics, thus making it much easier to read the statistics in VSD.

Set the cache name soon after login. If you are collecting statistics information using statmonitor, information may be logged using the default name for the Gem, and you may have two separate lines of data for the same session.

Session Statistics

In addition to the system-generated statistics listed below, GemStone provides a facility for defining session statistics — user-defined statistics that can be written and read by each session, to monitor and profile the internal operations specific to your application.

There are 48 session cache statistic slots available, with names of the form SessionStat01...SessionStat47.

You can use the following methods to read and write the session cache statistics:

`System class >> sessionCacheStatAt:` *anIndex*

Returns the value of the statistic at the designated index. *anIndex* must be in the range -2 to 47. Negative values are reserved for internal use.

`System class >> sessionCacheStatAt:` *anIndex* `put:` *aValue*

Assigns a value to the statistic at the designated index and returns the new value. *anIndex* must be in the range -2 to 47. Negative values are reserved for internal use.

`System class >> sessionCacheStatAt:` *anIndex* `incrementBy:` *anInt*

Increment the statistic at the designated index by *anInt*, and returns the new value. *anIndex* must be in the range -2 to 47. Negative values are reserved for internal use.

`System class >> sessionCacheStatAt:` *anIndex* `decrementBy:` *anInt*

Decrement the statistic at the designated index by *anInt*, and returns the new value. *anIndex* must be in the range -2 to 47. Negative values are reserved for internal use.

`System class >> sessionCacheStatsForProcessSlot:` *aProcessSlot*

Return an array containing the 48 session statistics for the given process slot, or nil if the process slot is not found or is not in use.

`System class >> sessionCacheStatsForSessionId:` *aSessionIdt*

Return an array containing the 48 session statistics for the given session id, or nil if the session is not found or is not in use.

Global Session Statistics

In addition to the Gem session statistics, GemStone/S 64 Bit provides global session statistics — user-defined statistics that can be written and read by any Gem on any Gem server. Unlike *session* cache statistics, which are stored in the shared page cache of the machine that the Gem is running on, *global* session statistics are stored in the shared page cache of the Stone. Global session statistics are not transactional. For a given statistic, every session sees the same value, regardless of its transactional view.

There are 48 global cache statistic slots available, with names of the form GlobalStat01...GlobalStat47.

You can use the following methods to read and write the global cache statistics:

`System class >> globalSessionStatAt:` *aProcessSlot*

Returns the value of the statistic at the designated slot (must be in the range 0..47).

`System class >> globalSessionStatAt:` *aProcessSlot* `put:` *aValue*

Assigns a value to the statistic at the designated slot (must be in the range 0..47) and returns the new value. The value must be a SmallInteger in the range of -2147483648 to 2147483647.

`System class >> incrementGlobalSessionStatAt:` *aProcessSlot* `by:` *anInt*

Increments the value of the statistic at the designated slot by *anInt* and returns the new value of the statistic. The value *anInt* must be a SmallInteger in the range of -2147483648 to 2147483647.

Host System statistics

Host CPU statistics for Gems

The statistics for UserTime and SysTime require an OS call, which can cause cache statistics to impact performance. These statistics are not part of the information returned by regular cache statistics interface methods. To get this information, use the following methods.

System class >> hostCpuStatsForProcessId: *anInt*
Return an Array of two integers as follows:

1 - user mode CPU milliseconds
2 - system mode CPU milliseconds

Both array elements will be -1 if the process slot is out of range or not in use or if this method is not supported for the host architecture.

It is not required that the process with pid *anInt* is attached to the shared page cache or even is a GemStone process. The method will succeed for any process for which the Gem session executing the method has permission to view the target process' CPU usage statistics.

System class >> hostCpuStatsForProcessSlot: *anInt*
For the process using the cache process slot *anInt*, return an Array of two integers as follows:

1 - user mode CPU milliseconds used
2 - system mode CPU milliseconds used

Both array elements are set to -1 if the process slot is out of range or not in use, or if this method is not supported for the host architecture.

OS level statistics

While most monitoring is of the object server and session processes, it is also useful to monitor the performance of the operating system that is running GemStone. On host platforms that support it, the following methods return statistics provided by the operating system. This is the same information that is available via statmonitor; for details, see page 413.

System class>> fetchSystemStatNames
Return an array of Strings with the names of the available OS level statistics. The length is host-dependent. If the host system does not support system statistics, this method returns nil.

System class >> fetchSystemStats
Return an array of Numbers corresponding to the names returned by he #fetchSystemStatNames method. The length of the result array is host dependent. While most elements in the result array will be SmallIntegers, the result may also contain other types of Numbers such as SmallDoubles, Floats, LargeIntegers, etc. If the host system does not support system statistics, this method returns nil.

Host CPU Usage

Monitoring CPU usage for the host can

`System class >> hostCpuUsage`

Returns an Array of 5 SmallIntegers with values between 0 and 100 which have the following meanings:

1 - Percent CPU active (user + system)
2 - Percent CPU idle
3 - Percent CPU user
4 - Percent CPU system (kernel)
5 - Percent CPU I/O wait

On hosts with multiple CPUs, these figure represent the average across all processors. The results of the first call to this method are invalid and should be discarded. Returns nil if the host system does not support collecting CPU statistics.

Chapter

6 User Accounts and Security

This chapter also shows you how to perform some common GemStone user administration tasks:

- How to create and modify user accounts, including passwords, privileges, group memberships, and symbol resolution, and how to control the user's read-write access to objects through the use of *object security policies*.
- How to set up alternate ways to authenticate logins, such as authenticating by UNIX userId or LDAP.
- How to configure GemStone login security by restricting valid passwords, imposing password and account age limits, and monitoring intrusion attempts.

To perform most of these tasks you must have explicit *privilege* to execute a restricted Smalltalk method, and you may also need to be explicitly authorized to modify an affected GsObjectSecurityPolicy. This chapter introduces these concepts. For a full description of privileges and GsObjectSecurityPolicies, see the chapter of the *GemStone/S 64 Bit Programming Guide* that discusses security.

6.1 GemStone Users

This section provides background information about how GemStone stores user accounts, what accounts are predefined, and what determines an account's name space.

UserProfiles

Each GemStone user is associated with an instance of class UserProfile. This UserProfile object contains information describing objects that the user is allowed to examine or modify, privileges that the user has to perform certain operations, and security information.

Each UserProfile has the following information:

User ID	A unique String that identifies the user to the GemStone system.
Password	The GemStone-specific password (an InvariantString) to use to validate logins. GemStone stores the password in encrypted form in a secure manner.
Default Object Security Policy	Either nil or an instance of GsObjectSecurityPolicy. This determines the default read and write authorizations for objects created by the user.
Privileges	A collection of symbols that allow the user to perform certain "privileged" system functions.
Groups	In conjunction with object security policies, group membership is used to allow access to restricted objects for specific categories of users.
SymbolList	The list of SymbolDictionaries that this user can see.

These are discussed in more detail starting on page 186.

Other information related to the user account is stored in an instance of UserSecurityData; this includes data related to security features. Instances of UserSecurityData are private and protected, but some information, such as lastLoginTime, can be accessed via methods in UserProfile.

AllUsers

Each instance of UserProfile must be in the global collection, AllUsers. AllUsers is the single instance of UserProfileSet. AllUsers acts as the "root" for all objects in the repository; any object in the repository must be reachable from AllUsers, usually via the SymbolLists of the UserProfiles, otherwise it is subject to garbage collection.

Special System Users

When GemStone is first installed, AllUsers has UserProfiles already defined for the following users. These are the special system users. *You must never delete these users.* These users may not have privileges removed, cannot be disabled, must use GemStone authentication, and their accounts are not subject to password or account age limits.

You can determine if an account is a special system user by executing:

```
UserProfile isSpecialUserId: 'theUserId'
```

SystemUser

SystemUser is analogous to `root` in UNIX. SystemUser has all privileges, belongs to all predefined groups, and is authorized to read and write all objects regardless of GsObjectSecurityPolicy protection. These privileges cannot be taken away, so SystemUser can always write to all objects. This account is used only to perform GemStone system upgrades, modify some system configuration settings, and other special-purpose operations that must be highly restricted.

The SystemUser account is the owner of the SystemObjectSecurityPolicy, which contains the kernel classes.

WARNING

Logging in to GemStone as SystemUser is like logging in to your workstation as root: an accidental modification to a kernel object can cause a great deal of harm. Use the DataCurator account for system administration functions except those that ***require*** *SystemUser privileges, such as a repository upgrade.*

DataCurator

The DataCurator account is the account that is normally used for day-to-day administration tasks. Initially, DataCurator is granted all privileges and belongs to all predefined groups. All GemStone UserProfiles are protected by the DataCuratorObjectSecurityPolicy.

GcUser

The GcUser account is a special account that logs in to the repository automatically to perform garbage collection tasks. You normally would only login as GcUser in order to update configuration parameters stored in GcUser's UserGlobals.

SymbolUser

The SymbolUser account is a special account that is used to perform symbol creation tasks. Login as SymbolUser is disallowed. Directly accessing the AllSymbols collection, which is in the UserGlobals of the SymbolUser, is possible from another administrative session.

Nameless

The Nameless account is a special account for use only by other GemStone products. Do not use this account or change it unless instructed to do so by GemStone Technical Support.

6.2 Creating and Removing Users

Methods that create UserProfiles add the new UserProfile to AllUsers, the global collection (UserProfileSet) of users. UserProfiles that are not in AllUsers cannot log in.

In addition to creating the new UserProfile, you should also see that each user's UNIX environment is set up to provide access to GemStone. This is described in the *GemStone/S 64 Bit Installation Guide*.

Removing a user, in addition to removing the UserProfile from AllUsers, requires cleanup to ensure that objects that refer to the deleted user do not inadvertently prevent the deleted user from being garbage collected, and that objects that are referred to only by the deleted users are not inadvertently garbage collected. The methods to remove users perform this cleanup.

UserProfile Data

Each user profile has the following instance variable data, either explicitly specified during instance creation, or provided with default values. This information can be updated.

The requirements for updating this information vary. In many cases, the requirements are different between updating information for another user and for updating your own information. See the update methods for details.

User ID

Each UserProfile is created with a unique user Id String. Embedded spaces are permitted.

UserId may only be changed by SystemUser, and the userIds of special system users cannot be changed.

Password

Each UserProfile is created with an initial password, an Invariant String, which must not be the same as the User Id and may not be longer than 1024 characters. The user supplies this password for identification purposes at login, unless another form of login authentication is used; see "Password Authentication" on page 209.

Users must have the explicit privilege #UserPassword to change their own passwords, and there are a number of ways to constrain the choice of passwords. See the section starting on page 214 for details.

To change the password of a user other than yourself, the #OtherPassword privilege is required, a different method is used, and password constraints do not apply.

Default Object Security Policy

A users' defaultObjectSecurityPolicy determines the default read and write authorizations for objects created by the user. When you add a new user to the GemStone system, you can either allow the default security policy to be nil, use protocol that creates a new GsObjectSecurityPolicy, or specify an existing GsObjectSecurityPolicy for the user.

For more information on how security policies are used to control read and write authorization for objects in the repository, see the chapter in the *GemStone/S 64 Bit Programming Guide* that discusses security.

A defaultObjectSecurityPolicy of nil means that objects created by that user, by default, have world read and write access; that is, are not restricted from being read or written by all other users. Not requiring authorization checks has the benefit of improved performance, if your application does not require object level security.

To specify a defaultObjectSecurityPolicy for a user, you may use an existing instance of GsObjectSecurityPolicy, or you must create and commit the GsObjectSecurityPolicy before it can be used.

To modify the defaultObjectSecurityPolicy of any user, you must have write access to the security policy of that UserProfile. In addition, to modify your own defaultObjectSecurityPolicy, you must have the #DefaultObjectSecurityPolicy privilege.

Privileges

When you create a new UserProfile, you determine whether the new user may perform certain "privileged" system functions. For example, stopping another session or the repository itself requires a particular privilege to do so. Table 6.1 describes the types of functions that each privilege controls.

For developers, you must also specifically grant the privilege that allows the user to modify code.

Note that privileges are more powerful than security policy authorization (see page 198). Although the owner of a security policy can always use authorization protocol to restrict read or write access to objects in a policy, an administrator with appropriate privileges, such as DataCurator, can override that protection by sending privileged messages that let you change the authorization scheme.

Table 6.1 GemStone Privileges

Type of Privilege	Privileged Operations
SystemControl	SystemControl is required by methods that start or stop sessions, including operations that invoke the multi-threaded scan (see page 339); for methods that suspend or resume logins, send signals to other sessions, and manage checkpoints.
SessionAccess	SessionAccess privilege is required to find out information about sessions other than the current session, or to perform operations on other sessions.
UserPassword	Required to change your own password using `UserProfile>>oldPassword:newPassword:`
DefaultObjectSecurityPolicy	This privilege is required to set a UserProfile's default ObjectSecurityPolicy using `UserProfile>>defaultObjectSecurityPolicy:` or a method that invokes that. For compatibility with previous versions, the DefaultSegment privilege also resolves to this privilege

Table 6.1 GemStone Privileges (Continued)

Type of Privilege	Privileged Operations
CodeModification	You must have CodeModification privilege to create or modify instances of GsNMethod, GsMethodDictionary, or Class. Also see the discussion following this table.
OtherPassword	You must have OtherPassword privilege to make any changes to a UserProfile. This includes adding or removing a SymbolDictionary to/from a SymbolList that is not your own. OtherPassword is also required to find out information about UserProfiles other than the currently logged in session. OtherPassword is required to make any changes to AllUsers, including creating a new user and configuring security requirements.
ObjectSecurityPolicyCreation	Required in order to creating a new GsObjectSecurityPolicy, using `GsObjectSecurityPolicy class>>newInRepository:` or any methods that invoke this. For compatibility with previous versions, the SegmentCreation privilege also resolves to this privilege.
ObjectSecurityPolicyProtection	You must have ObjectSecurityPolicyProtection to update the authorizations of a GsObjectSecurityPolicy, other than one that is owned by the current session's user. This includes `GsObjectSecurityPolicy>>group:authorization:`, `ownerAuthorization:`, and `worldAuthorization:`. For compatibility with previous versions, the SegmentProtection privilege also resolves to this privilege.
FileControl	FileControl is required for operations that access external files, including operations related to backup, restore, transaction logs, and extents.

Table 6.1 GemStone Privileges (Continued)

Type of Privilege	Privileged Operations
GarbageCollection	Required to perform any garbage collection operation, to start and stop Admin and Reclaim GcGems, and force epoch or reclaim to run. Also required to audit and profile the repository.
NoPerformOnServer	If you have this privilege, you cannot execute `System class>>performOnServer:`
NoUserAction	If you have this privilege, you cannot execute `System class>> loadUserActionLibrary:`
NoGsFileOnServer	If you have this privilege, you cannot execute ny GsFile operation which accesses a file on the server.
NoGsFileOnClient	If you have this privilege, you cannot execute any GsFile operation which accesses a file on the client.
SessionPriority	Required to modify the priority of any session, or to check the priority of a session other than the current session.

Code Modification Privilege

CodeModification privilege is required to execute any method that modifies Smalltalk code:

- You must have CodeModification privilege to create instances of GsNMethod, or to create or modify instances of GsMethodDictionary or Class. (You cannot modify a GsNMethod once it has been created.)
- You must have CodeModification privilege to add a Class to, or remove a Class from, a SymbolDictionary and its subclasses.
- You must have CodeModification privilege to add or remove a SymbolDictionary from your own SymbolList. (See page 204)

You cannot use GemBuilder for C to modify instances of the following classes (or their subclasses): GsNMethod, GsMethodDictionary, Class, SymbolDictionary, SymbolList, UserProfile.

Groups

GemStone uses group membership to facilitate access to objects that are protected by GsObjectSecurityPolicies. While certain objects must be protected from read or write access by other users in the system (the "world"), you may still need to grant access to specific individual users. By adding group authorization to the

GsObjectSecurityPolicy that protects these objects, and adding the corresponding group membership to that user, you can provide a user with the appropriate access to these objects.

A GsObjectSecurityPolicy can authorize multiple groups and a user can be a member of multiple groups. There are several predefined groups, as shown in Table 6.2. By default, all new users become members of group Subscribers.

AllGroups

AllGroups is a global collection of Strings, that includes all groups defined for all users and all security policies.

Initially, AllGroups contains the following:

Table 6.2 GemStone Groups

Group name	Access
System	Members of this group have write access to objects protected by the GcUser's object security policy.
DataCuratorGroup	Members of this groups have write access to objects protected by the **DataCuratorObjectSecurityPolicy**. This is useful if you wish to make a user other than DataCurator to be a system administrator, since many operations that update users require write access to DataCuratorObjectSecurityPolicy.
Publishers	Members of this group have write access to objects protected by **PublishedObjectSecurityPolicy**.
Subscribers	Members of this group have read access to objects protected by **PublishedObjectSecurityPolicy**.
SymbolUser	(Reserved).

Symbol Lists

As explained in the *GemStone/S 64 Bit Programming Guide*, the GemStone Smalltalk compiler follows a well-defined path in resolving objects named by source code symbols. First, the compiler considers the possibility that a variable name might be either local (a temporary variable or an argument) or defined by the class of the current method definition (an instance variable, a class variable, or a pool variable). If a variable is none of these, the compiler refers to an Array of SymbolDictionaries in the user's UserProfile and current session state. That Array is called the user's *symbol list*. The symbol list tells Smalltalk which of many

possible GemStone SymbolDictionaries to search for an object named in a Smalltalk program.

For each session, a persistent instance of class SymbolList is stored in the repository and is referenced from the UserProfile associated with this session as the symbolList instance variable. In addition, a transient copy of that SymbolList is stored in the GsCurrentSession object for the logged-in session.

A session's transient copy can be modified without affecting (or causing concurrency conflicts with) either the persistent symbol list or the transient copies controlling other sessions. Changes to your own UserProfile's persistent symbol list also change the symbol resolution of your current session. However, changes to the persistent symbol list are likely to cause concurrency conflicts with other sessions logged in under the same userId.

For further information about symbol lists, refer to the *GemStone/S 64 Bit Programming Guide*.

New UserProfiles are created with the following SymbolDictionaries, in this order:

UserGlobals Each UserProfile has its own SymbolDictionary for the user's private symbols.

Globals The second element in each user's initial symbol list is a "system globals" SymbolDictionary, *Globals*. This dictionary contains all of the GemStone Smalltalk kernel classes (Object, Class, Collection, Integer, and so forth). Although users can read the objects in Globals, ordinarily they cannot modify objects in that Dictionary.

Published The third and final element in each user's initial symbol list is a SymbolDictionary for application objects that are "published" to all users. Users who are members of the group Publishers can place objects in this dictionary to make them visible to other users. Using the Published dictionary lets you share these objects without having to put them in Globals, which contains the GemStone kernel classes, and without the necessity of adding a special dictionary to each user's symbolList instance variable.

Although all users automatically share access to objects in Globals, sharing application objects between users requires that the objects be in a SymbolList that is visible to both users. There are three primary ways to do this:

- As a member of group Publishers, you can add the objects to the Published dictionary. This dictionary is already in each user's symbol list, so whatever you add becomes visible to users the next time they obtain a fresh transaction

view of the repository. You may do this by sending the message `Published at:` *aKey* `put:` *aValue*.

- You can define a special SymbolDictionary, and add that to the user's SymbolList. The procedure is described under ""Adding a SymbolDictionary to Someone Else's Symbol List" on page 205".
- The application itself can add the objects to the individual user's symbol list, either to the permanent symbol list in the UserProfile or to a transient symbol list for that session. For information about this approach, refer to the *GemStone/S 64 Bit Programming Guide*.

For more information, refer to the chapter on symbol resolution and object sharing in the *GemStone/S 64 Bit Programming Guide*.

Creating Users

Privileges required: OtherPassword, and write access to the DataCuratorObjectSecurityPolicy. You may also need ObjectSecurityPolicyCreation.

Simple User Creation

At minimum to create a new UserProfile, you must supply the new user's userId and password, each as a String. This creates the new user with no privileges, only the Subscribers group, and with a nil defaultObjectSecurityPolicy.

```
AllUsers addNewUserWithId: 'theUserId'
   password: 'thePassword'.
```

Simple User Creation with GsObjectSecurityPolicy Creation

To create a new user and specify that a new instance of GsObjectSecurityPolicy should be created for the new user, use the following method:

```
AllUsers addNewUserWithId: 'theUserId'
   password: 'thePassword'
   createNewObjectSecurityPolicy: true
```

User Creation With Privileges, Groups, and ObjectSecurityPolicy

Using the complete form allows you to assign privileges to the new user, add the user to groups, and specify an ObjectSecurityPolicy. The ObjectSecurityPolicy may be nil.

```
AllUsers addNewUserWithId: 'theUserId'
    password: 'thePassword'
    defaultObjectSecurityPolicy: anObjectSecurityPolicyOrNil
    privileges: anArrayOfPrivStrings
    inGroups: aCollectionOfGroupStrings
```

For example:

```
topaz 1> printit
AllUsers addNewUserWithId: 'Mary'
   password: 'herPasswd'
   defaultObjectSecurityPolicy: nil
   privileges: #( 'UserPassword' )
   inGroups: #( 'MarathonRunners' ).
"commit the new UserProfile"
System commitTransaction.
%
```

For additional user creation protocol, see the image. UserProfileSet instance methods and UserProfile class methods both create new UserProfiles and add the new user to AllUsers.

Removing Users

Privileges required: OtherPassword.

In addition to removing the UserProfile from AllUsers, removing a user requires that any GsObjectSecurityPolicies owned by the deleted user are moved to a new user.

In addition, to ensure that work being done by the user being deleted is not lost, the SymbolLists of the deleted user are moved to a separate location where they can be periodically reviewed and manually dereferenced.

DeletedUserProfile and AllDeletedUsers

When a user is removed, a new instance of DeletedUserProfile is created, with the userId and the SymbolLists of the user being removed, and the current DateTime.

This instance of DeletedUserProfile is moved to a collection #AllDeletedUsers, in Globals SymbolDictionary.

An administrator should review the classes and methods in the SymbolLists of the DeletedUserProfile, copy anything valuable elsewhere, and manually remove the DeletedUserProfile from AllDeletedUsers.

Remove User, with ObjectSecurityPolicies Going to the Current User

The following methods remove the user and reassign any GsObjectSecurityPolicies owned by the user to be removed to the UserProfile of the current user (the user executing this code, such as DataCurator).

This form passes in the instance of UserProfile for the user to be deleted:

```
AllUsers removeAndCleanup: aUserProfile.
```

While this form passes in the userId of the user to be deleted, and includes a block to execute if no UserProfile with the given userId exists in AllUsers:

```
AllUsers removeAndCleanupUserWithId: 'aUserId'
   ifAbsent: aBlock
```

Remove User, with ObjectSecurityPolicies going to another User

The following methods remove the user and reassign any GsObjectSecurityPolicies owned by that user to another specific UserProfile.

This form passes in the instance of UserProfile for the user to be deleted and the UserProfile to which to reassign the ObjectSecurityPolicies:

```
AllUsers removeAndCleanup: aUserProfile
   migrationSecurityPoliciesTo: anotherUserProfile.
```

While this form passes in the userId String of the user to be deleted and the userId String of the UserProfile to which to reassign the ObjectSecurityPolicies, and includes a block to execute if a UserProfile does not exist with either of the UserId Strings:

```
AllUsers removeAndCleanupUserWithId: 'aUserId'
   migrationSecurityPoliciesUserWithId: 'anotherUserId'
   ifAnyAbsent: aBlock
```

For example,

```
topaz 1> printit
AllUsers removeAndCleanup: (AllUsers userWithId: 'John')
   migrationSecurityPoliciesTo: (AllUsers userWithId: 'Ann')
System commitTransaction.
%
```

6.3 Administering Users

List Existing Users

Privileges required: None.

There is no direct method within GemStone Smalltalk to list only the names of existing accounts. The following example shows one way to obtain that information:

```
topaz 1> printit
(AllUsers collect: [:each | each userId ]) asArray.
%
#1 DataCurator
#2 SystemUser
#3 SymbolUser
#4 GcUser
#5 Nameless
```

Modifying the UserId

You must be SystemUser to update a User's UserId.

The new user ID will take effect when you commit the current transaction.

To modify the user ID of a GemStone user, as SystemUser, execute the following expression:

```
topaz 1> printit
(AllUsers userWithId: 'theUserId') userId: 'newId'.
%
```

An error is raised if *newId* is the userId of an existing UserProfile.

Modifying Password

Users Changing Their Own Password

Privileges required: UserPassword.

In many cases, users set their own passwords and may be required to update them periodically. These users must be given the UserPassword privilege to do so, and use the following method to update their password:

```
UserProfile >> oldPassword:'oldPwd' newPassword: 'newPwd'
```

For example,

```
topaz 1> printit
System myUserProfile
   oldPassword: 'oldPasswordString'
   newPassword: 'newPasswordString'.
System commitTransaction
%
```

Password choice is constrained by login security that is configured for the repository; see the discussion on password restrictions starting on page 214.

A different method, requiring other privileges, is used by Administrators to update the password of another user.

The new password takes effect when you commit the current transaction.

Changing Another User's Password

Privileges required: OtherPassword.

To modify the password of any GemStone user, execute the following expression.

```
topaz 1> printit
(AllUsers userWithId: 'theUserId')
   password: 'newPasswordString' .
System commitTransaction
%
```

The new password takes effect when you commit the current transaction.

The password set by this method is not subject to the constraints described on page 214 because it can only be set by a user having the OtherPassword privilege. The password must not be the same as the UserId and must not be longer than 1024 characters.

Each password change of this type is noted in the GemStone security log, which currently is the Stone's log file. The entry includes the userId of the session making the change but not the new password.

Modifying defaultObjectSecurityPolicy

Each security policy maintains access authorization for its owner, the world, and an unlimited number of groups. There are three levels of authorization: none, read (read-only), and write (which includes read permission).

Determining Who Is Authorized to Read or Write in an Object Security Policy

Privileges required: read authorization for the security policy with which this policy is associated, such as the DataCuratorObjectSecurityPolicy.

You can find out who is authorized to read or write objects in an security policy by sending it the message `asString`. For instance:

```
topaz 1> printit
PublishedObjectSecurityPolicy asString
%
anObjectSecurityPolicy, Number 6 in Repository
SystemRepository, Owner SystemUser write, Group Subscribers
read, Group Publishers write, World none
```

Changing the Authorization of an Object Security Policy

Privileges required: ObjectSecurityPolicyProtection or be the security policy's owner, and write authorization to the DataCuratorObjectSecurityPolicy.

The new authorization will take effect for logins following the commit of the current transaction.

CAUTION

Do not attempt to change the authorization of SystemObjectSecurityPolicy.

To change the authorization for a security policy, execute any (or all) of the following expressions.

```
topaz 1> printit
theObjectSecurityPolicy ownerAuthorization: #anAuthorizationSymbol .
%
topaz 1> printit
theObjectSecurityPolicy worldAuthorization: #anAuthorizationSymbol .
%
topaz 1> printit
theObjectSecurityPolicy group: 'aGroupString'
   authorization: #anAuthorizationSymbol .
%
```

NOTE

Exercise caution when changing the authorization for any security policy that a user may be using as his or her default or current security

policy — whether or not the user owns the affected policy. If a user attempts to commit a transaction, but has created objects with a policy for which he or she no longer has write authorization, an error will be generated.

For example, to authorize the group Accounting to read (but not write) in user Eli's default security policy, you could execute the following expression:

```
topaz 1> printit
(AllUsers userWithId: 'Eli') defaultObjectSecurityPolicy
   group: 'Accounting'
   authorization: #read.
%
```

If the group `'Accounting'` does not exist, GemStone will return an error. The discussion under "Adding a User to a Group" on page 201"explains how to create a new GemStone group.

Remove a Group from an Object Security Policy's Authorization List

Privileges required: ObjectSecurityPolicyProtection, and write authorization for the security policy.

To remove a group from a security policy's list of authorized groups, execute the following expression:

theSecurityPolicy **group:** *'aGroupString'* **authorization: #none**

Change a User's Default Object Security Policy

Privileges required: DefaultObjectSecurityPolicy to change your own, or write authorization in the DataCurator ObjectSecurityPolicy to change another's; and write authorization to the DataCurator ObjectSecurityPolicy.

Changes to another user's default security policy do not take effect until the next login.

To change a user's default security policy, execute the following expression:

(AllUsers userWithId: *'theUserId'***)**
defaultObjectSecurityPolicy: *aNewSecurityPolicy*

NOTE
If you change any user's default security policy (including your own) to

a security policy for which that user lacks write authorization, and you subsequently commit the transaction, the affected user will no longer be able to log in to GemStone.

Modifying Groups

Examining Group Memberships

No privileges are required for this operation.

To find out which groups a user belongs to, execute the following expression:

```
(AllUsers userWithId: 'theUserId') groups
```

This expression returns a Set of Strings indicating the groups to which the user belongs.

Adding a User to a Group

Privileges required: OtherPassword and write authorization to DataCuratorObjectSecurityPolicy.

To add a user to a group, do the following:

```
topaz 1> printit
" If this is a new group, add it to AllGroups "
('MarathonRunners' in: AllGroups)
   ifFalse: [AllGroups add: 'MarathonRunners' ].
(AllUsers userWithId: 'theUserId') addGroup: 'MarathonRunners'.
System commitTransaction
%
```

This expression adds the user to the group MarathonRunners by adding the group name to the list of groups maintained in the UserProfile. (This action takes effect when you commit the current transaction.)

If the group MarathonRunners did not previously exist, this expression creates it in AllGroups (the "master list" of all group names). See page 191 for more information about AllGroups.

Removing a User from a Group

Privileges required: OtherPassword and write authorization to DataCuratorObjectSecurityPolicy.

To remove a user from a group, execute an expression of the following form:

```
topaz 1> printit
(AllUsers userWithId: 'theUserId') removeGroup: 'Sprinters' .
System commitTransaction
%
```

This expression removes the designated group from the list of groups to which the user belongs. This action will take effect when you commit the current transaction. For more information about groups, see the Security chapter of the *GemStone/S 64 Bit Programming Guide*.

Listing Members of a Group

No privileges are required for this operation.

To list all members of a user group, execute the following expression:

```
AllUsers membersOfGroup: aString
```

This expression returns an IdentitySet containing the userId for each member of the group.

Removing a Group

Privileges required: OtherPassword and write authorization to DataCuratorObjectSecurityPolicy.

To remove a user group from the global object AllGroups, execute the following expression.

```
topaz 1> printit
| theGroup |
theGroup := 'aGroupString'.
" Does the group still contain any members? If so, first
remove each member from the group "
(AllUsers usersInGroup: theGroup) do:
        [:aUserProfile| aUserProfile removeGroup: theGroup].
" It's OK to remove the group itself "
AllGroups remove: theGroup.
System commitTransaction

%
```

Modifying Privileges

Examining a User's Privileges

No privileges are required for this operation.

GemStone provides messages that allow you to determine which privileged methods a GemStone user may execute, and to change the privileges of any user. Naturally, you need the appropriate privileges to use those methods.

To find out which privileged methods a given user is permitted to execute, send the following message to the desired user's UserProfile:

```
topaz 1> printit
(AllUsers userWithId: 'theUserId') privileges
%
```

This message returns an Array of Strings. Each String in the Array corresponds to one of the user's privileges. Table 6.1 (on page 188) lists the Smalltalk methods that correspond to each privilege.

Adding a Privilege

Privileges required: OtherPassword and write authorization to the ObjectSecurityPolicy of the user's UserProfile.

The new privileges will take effect when you commit the current transaction.

To add to a user's existing privileges, execute the following expression:

```
(AllUsers userWithId: 'theUserId')
   addPrivilege: aPrivilegeString .
```

Here's an example that assigns three new privileges to user Bob:

```
(AllUsers userWithId: 'Bob')
   addPrivilege: 'SystemControl';
   addPrivilege: 'SessionAccess';
   addPrivilege: 'UserPassword' .
System commitTransaction
%
```

Revoking a Privilege

Privileges required: OtherPassword and write authorization to the security policy of the user's UserProfile.

The privileges will be revoked when you commit the current transaction.

To revoke one (or more) of a user's existing privileges, execute the following expression:

```
(AllUsers userWithId: 'theUserId')
   deletePrivilege: aPrivilegeString .
```

The following example revokes three of user Jane's privileges:

```
(AllUsers userWithId: 'Jane')
   deletePrivilege: 'SystemControl';
   deletePrivilege: 'SessionAccess';
   deletePrivilege: 'UserPassword' .
System commitTransaction
%
```

Reassigning All Privileges

Privileges required: OtherPassword and write authorization to the security policy of the user's UserProfile.

The new privileges will take effect when you commit the current transaction.

To redefine the full set of a user's privileges, perhaps adding some and revoking others, execute the following expression:

```
(AllUsers userWithId: theUserId) privileges: anArrayOfStrings
```

This expression supersedes any previous privilege assignments. After the change is committed, only those privileges listed in the expression are valid for the user. Any privileges that were previously valid, but are not listed, are revoked.

For example:

```
topaz 1> printit
(AllUsers userWithId: 'Sam') privileges:
   #( 'UserPassword' ) .
System commitTransaction

%
```

Modifying SymbolLists

Adding a SymbolDictionary to Your Own Symbol List

Privileges required: CodeModification.

You can add a dictionary to a symbol list by sending the message `UserProfile>>insertDictionary:` *aSymbolDictionary* `at:` *anIndex*. The change does not affect the transient copy of the symbol list that is used by another currently logged in session until that session commits or aborts.

This example inserts dictionary NewDict (which already exists in the Published dictionary) into the user's own symbol list:

```
System myUserProfile
   insertDictionary: NewDict at: 2.
```

Inserting the new dictionary at index 2, as in the example, places it between the UserGlobals and the Globals dictionaries in the search order. Because symbol resolution depends on the order of dictionaries in a user's symbol list, the index used in this example may not be appropriate for all situations.

Adding a SymbolDictionary to Someone Else's Symbol List

Privileges required: OtherPassword and write permission to the security policy of the other user's SymbolDictionary.

This example inserts dictionary NewDict (which already exists in the Published dictionary) into user Jerry's symbol list:

```
topaz 1> printit
(AllUsers userWithId: 'Jerry')
   insertDictionary: NewDict at: 7.
System commitTransaction
%
```

Removing a SymbolDictionary from Your Own Symbol List

Privileges required: CodeModification.

You can remove a dictionary from a symbol list by sending the message `UserProfile>>removeDictionary:` *aSymbolDictionary*.

The change does not affect the transient copy of the symbol list that is used by another currently logged in session, until that session commits or aborts.

This example removes dictionary OldDict from the user's own symbol list:

```
topaz 1> printit
System myUserProfile
   removeDictionary: OldDict.
System commitTransaction
%
```

Removing a SymbolDictionary from Someone Else's Symbol List

Privileges required: OtherPassword and write permission to the security policy of the other user's SymbolDictionary,

This example removes dictionary OldDict from user Jerry's symbol list:

```
topaz 1> printit
(AllUsers userWithId: 'Jerry')
   removeDictionary: OldDict.
System commitTransaction
%
```

Disable and Enable User Logins

A disabled account cannot log in. Disabling an account consists of setting its password to a non-enterable character. Once an account is disabled, a new password must be assigned for that account by an administrator, before the account can log in again.

Disable an Account Explicitly

Privileges required: OtherPassword. Logins cannot be disabled for special system accounts.

Users can be disabled using the method `disable` or `disableWithReason:`. This prevents the user from logging in, but does not affect current login sessions.

For example:

```
topaz 1> printit
(AllUsers userWithId: 'Sam')
   disableWithReason: 'on Sabbatical'.
System commitTransaction

%
```

Re-enable an Account

Privileges required: OtherPassword.

To re-enable a user account, an administrative user must login and reset the users's password. For example, to reset Sam's password and require him to change his password the first time he logs in:

```
topaz 1> printit
(AllUsers userWithId: 'Sam')
   password: 'AaBbCc';
   loginsAllowedBeforeExpiration: 1.
System commitTransaction
```

Find Out Which Accounts Have Been Disabled

Privileges required: OtherPassword.

The message `AllUsers findDisabledUsers` returns a SortedCollection of UserProfiles that are disabled explicitly or by one of the security precautions discussed in this chapter:

- The password expired (through aging or a login limit).
- The account remained inactive.
- There were repeated password failures.

For example:

```
topaz 1> level 1
topaz 1> printit
AllUsers findDisabledUsers
  collect: [:aUser | aUser userId ] .
%
an Array
  #1 qa2
  #2 qa3
```

DataCurator or another user with the OtherPassword privilege can reactivate an account by giving it a new password, as described on page 206.

Check If an Account Is Disabled

Privileges required: OtherPassword.

You can check if a particular account is disabled by sending the message `isDisabled` to the account's UserProfile. The method returns either True or False. This example inquires about account *qa2*:

```
topaz 1> printit
(AllUsers userWithId: 'qa2') isDisabled
%
true
```

Find Out Why an Account Was Disabled

Privileges required: OtherPassword.

You can find out why a particular account was disabled by sending the message `reasonForDisabledAccount` to the account's UserProfile. This example inquires about account *qa2*:

```
topaz 1> printit
(AllUsers userWithId: 'qa2') reasonForDisabledAccount
%
LoginsWithSamePassword
```

If the account was disabled using the method **`disableWithReason:`**, this method will return that argument. Otherwise if the account was disabled by login security, it will return one of these Strings: 'PasswordAgeLimit', 'StaleAccount', 'LoginsWithSamePassword', or 'LoginsWithInvalidPassword'.

Disable and Enable Commits by User

Commits can be disabled for particular users to ensure "read only" access to the GemStone repository. These users can still log in and view data for which they have read or write authorization, and can modify objects, but they cannot commit and make any changes permanent.

Disable Commits

Privileges required: OtherPassword. Commits cannot be disabled for special system users.

To disable commits for a user, execute the following expression:

`(AllUsers userWithId:` *theUserId*`) disableCommits`

This expression disables any commits for the given user, beginning with the next login of the user after the session making this change commits. If this user is currently logged in, It does not affect the user's transaction, if any.

For example:

```
topaz 1> printit
(AllUsers userWithId: 'Sam') disableCommits.
System commitTransaction
%
```

Re-enable Commits for a User

Privileges required: OtherPassword.

To enable commits for a user, execute the following expression:

(AllUsers userWithId: *theUserId*) **enableCommits**

This expression enables commits for the given user, beginning with the next login after the session making this change commits.

Check If a User Can Commit

Privileges required: OtherPassword.

To test if commits have been disabled for a user account, execute the following expression:

(AllUsers userWithId: *theUserId*) **isReadOnly**

6.4 Password Authentication

When a user wants to log in to the GemStone repository, their login must be authenticated - they must present a password that is matched against stored information for their UserId, to verify that they are authorized to log in. This authentication can be done entirely within GemStone, or GemStone can use UNIX or LDAP to perform the authentication.

Performing the authentication entirely with GemStone - Gemstone authentication - is the initial default for all users. Other authentication schemes can be configured for individual UserProfiles, although special system users will always use GemStone authentication. The repository may contain UserProfiles using a mix of authentication schemes.

After the authentication scheme is modified for a user and the change is committed, it will take effect the next time the user logs in. Existing logins are not affected.

GemStone authentication

Privileges required: OtherPassword

GemStone authentication is the default authentication, using the UserId and password store with the UserProfile of the account. In older versions, this was the only way of authentication within GemStone, and must still always be used by the special system users - SystemUser, DataCurator, GcUser, SymbolUser, and Nameless.

```
topaz 1> printit
(AllUsers userWithId: 'Mary') enableGemStoneAuthentication.
System commitTransaction.
%
```

UNIX Authentication

Privileges required: OtherPassword

When UNIX account authentication is enabled for a user, they will enter their UNIX account password instead of their GemStone password. Password management for this user then becomes the Unix password management; sending messages to change the GemStone password are not useful, and these accounts are not subject to GemStone's password aging mechanisms.

The GemStone userId may be the same as the UNIX userId, or they may be different. When you enable UNIX authentication for an account, you may specify the UNIX userId associated with the GemStone UserProfile and this will be used for authentication. Using nil means to use the GemStone userId as the UNIX userId.

```
(AllUsers userWithId: GemStoneUserId)
   enableUnixAuthenticationWithAlias: UNIXUserIdOrNil
```

for example, if Mary's UNIX userId is msmith, you can use the first form if her GemStone userId is Mary. If her GemStone userId is also msmith you leave the argument nil.

```
topaz 1> printit
(AllUsers userWithId: 'Mary')
   enableUnixAuthenticationWithAlias: 'msmith'.
System commitTransaction.
%
topaz 1> printit
(AllUsers userWithId: 'msmith')
   enableUnixAuthenticationWithAlias: nil.
System commitTransaction.
%
```

To verify that the UNIX userId exists, execute:

System unixUserIdExists: *UNIXUserId*

LDAP Authentication

Privileges required: OtherPassword

An LDAP (Lightweight Directory Access Protocol) server consolidates and centralizes user authentication, and can be used to authenticate logins to the GemStone server.

To use LDAP for GemStone authentication, you must have an LDAP server available. When a UserProfile has been configured for LDAP authentication, on login, GemStone performs an LDAP bind to authenticate the userId.

In most cases, the LDAP bind requires a Distinguished Name (DN), which is the unique identifier for an entry. The DN includes both the userId and domain information, for example:

```
'uid=msmith,ou=employees,dc=somecompany,dc=com'
```

GemStone composes the DN based on the arguments provided when you configure LDAP authentication.

To configure a user to use LDAP for authentication, use the following method:

(AllUsers userWithId: *GemStoneUserId***)**
   enableUnixAuthenticationWithAlias: *UNIXUserIdOrNil*
   baseDn: *baseDn*
   filterDn: *filterDn*

As for UNIX authentication, if the LDAP userId is the same as the GemStone Id, you may pass in nil for this argument; otherwise, pass in the LDAP userId. The User will use their GemStone userId and the password for their LDAP account to log in.

Fully Qualified DN

You can configure the LDAP authentication for a particular user with the specific DN, with '%s' replacing the userId, for the baseDn: argument. In this case you should pass in nil for the filterDn: argument, which will disable the query.

For example, to configure authentication to use a specific fully qualified DN:

```
topaz 1> printit
(AllUsers userWithId: 'Mary')
   enableUnixAuthenticationWithAlias: 'msmith'
   baseDn: 'uid=%s,ou=employees,dc=somecompany,dc=com'
   filterDn: nil.
System commitTransaction.
%
```

Search for DN

You can also configure authentication to include the base domain information in the baseDn: argument, and include a filter in the filterDn: argument. The filter must contain '%s' in the position for the userId. GemStone will perform a query to get the full DN given the base and filter information.

The base and filter information is provided at the time the authentication is configured, not at login time, so a search option is particularly useful if the LDAP structure is likely to be modified.

To configure authentication to do a search for the given user:

```
topaz 1> printit
(AllUsers userWithId: 'Mary')
   enableUnixAuthenticationWithAlias: 'msmith'
   baseDn: 'dc=somecompany,dc=com'
   filterDn: '(uid=%s)'.
System commitTransaction.
%
```

Determining an Account's Authentication Scheme

Privileges required: OtherPassword

Your repository may contain a mix of authentication schemes, with some users (at least certainly the special system accounts) using GemStone authentication, others authenticating using UNIX accounts, and still other using LDAP.

You can determine what scheme an account is using by sending **authenticationScheme**. This will return the symbol #GemStone, #UNIX, or #LDAP. For example,

```
topaz 1> printit
(AllUsers userWithId: 'DataCurator') authenticationScheme.
%
GemStone
```

You may also send messages to check for specific schemes. The following methods return true or false.

```
(AllUsers userWithId: userId) authenticationSchemeIsUNIX

(AllUsers userWithId: userId) authenticationSchemeIsLDAP

(AllUsers userWithId: userId) authenticationSchemeIsGemStone
```

6.5 Configuring GemStone Login Security

GemStone provides several login security features. You can:

- Constrain the choice of passwords to a certain pattern, ban particular passwords altogether, or ban reuse of a password by the same account
- Require users to change their passwords periodically (password aging)
- Limit the number of logins under a temporary password
- Disable accounts that have not logged in for a specified interval (account aging)
- Limit the number of concurrent sessions by a particular account
- Monitor failed login attempts and, if necessary, disable further login attempts on that account

In all cases, the password must not be the same as the UserId and must not be longer than 1024 characters.

Additional methods let you determine which accounts have been disabled by one of these security features and why a particular account was disabled.

CAUTION
GemStone records certain administrative changes to these security features in the Stone log. You may want to restrict access to that file.

The special system users - SystemUser, DataCurator, SymbolUser, GcUser, and Nameless - are never disabled by the security features.

Limiting Choice of Passwords

Privileges required: OtherPassword and write authorization to DataCuratorObjectSecurityPolicy.

You can constrain a user's choice of passwords in terms of pattern (such as the number of characters that repeat). Independently, you can establish a list of words that are disallowed as passwords, and you can keep a user from choosing the same password more than once.

The constraints described here apply only when a user changes his or her own password by using the message `UserProfile>>oldPassword:newPassword:` and only to password changes after the constraint is committed to the repository. That is, the constraints (other than the prohibition of userId as the password) do not apply to administrator

actions changing any other account's password using the OtherPassword privilege, and they do not invalidate existing passwords.

Table 6.3 lists the messages that you can use to set pattern constraints. You send these messages to the global object AllUsers. For example, to set the minimum password length to six characters, do this:

```
topaz 1> printit
AllUsers minPasswordSize: 6 .
System commitTransaction
%
```

The default setting in all cases is 0, which means there is no constraint on the pattern.

Table 6.3 Ways to Constrain the Password Pattern

Message to AllUsers	Comments
`minPasswordSize:` *aPositiveInteger*	Sets the minimum number of characters in a new password; 0 means no constraint.
`maxPasswordSize:` *aPositiveInteger*	Sets the maximum number of characters in a new password; 0 disables the constraint. (The password String itself must not be longer than 1024 characters.)
`maxRepeatingChars:` *aPositiveInteger*	Sets the maximum number of adjacent characters that can have the same value; for example, *1* allows 'aba' but not 'aa'. 0 means no constraint.
`maxConsecutiveChars:` *aPositiveInteger*	Sets the maximum number of adjacent characters that can be an ascending or descending sequence, such as '123' or 'zyx' based on a case-sensitive comparison. 0 means no constraint.
`maxCharsOfSameType:` *aPositiveInteger*	Sets the maximum number of adjacent characters that can be of the same type (alpha, numeric, or special); for example, *3* allows 'abc4de' but not 'abcde'. 0 means no constraint.
`passwordRequiresUppercase:` *aBoolean*	Set the requirement that the password contain at least one uppercase character. false means no constraint.
`passwordRequiresLowercase:` *aBoolean*	Set the requirement that the password contain at least one lowercase character. false means no constraint.

Table 6.3 Ways to Constrain the Password Pattern

Message to AllUsers	Comments
`passwordRequiresSymbol:` *aBoolean*	Set the requirement that the password contain at least one character that is not alphanumeric. false means no constraint.
`passwordRequiresDigit:` *aBoolean*	Set the requirement that the password contain at least one digit. false means no constraint.

Any user can inquire about the current setting of a password pattern constraint by sending its corresponding Accessing message (that is, without the colon or argument shown in Table 6.3).

For example, to determine the current minimum size for a password:

```
topaz 1> printit
AllUsers minPasswordSize
%
6
```

Disallowing Particular Passwords

Privileges required: OtherPassword and write authorization to the DataCuratorObjectSecurityPolicy.

You can create a list of disallowed passwords by adding Strings to the AllUsers instance variable disallowedPasswords. Any messages understood by class Set can be used. For instance:

```
topaz 1> printit
(AllUsers disallowedPasswords)
  addAll: #( 'Mother' 'apple_pie' ) .
System commitTransaction
%
```

The default is an empty set.

Additions to this list affect only new passwords requested after the additions are committed; that is, additions do not invalidate existing passwords. If a user attempts to change that account's password to one of the Strings in disallowedPasswords, a SecurityError is signaled.

Any user can examine the current list of globally disallowed passwords by sending the message `AllUsers disallowedPasswords`.

Disallowing Reuse of Passwords

Privileges required: OtherPassword and write authorization to DataCuratorObjectSecurityPolicy.

You can prevent each user from choosing the same password more than once by setting AllUsers disallowUsedPasswords to true. By default, disallowUsedPasswords is false.

When reuse of passwords is disallowed, GemStone maintains a separate encrypted set of old passwords for each user. Each time a user invokes `oldPassword:newPassword:`, the new password is checked against the prior passwords for that account. If the new password matches a prior one, a SecurityError is signaled.

Disallow All Previously Used Passwords

To disallow password reuse, use the method:

`AllUsers disallowUsedPasswords:` *aBoolean*`.`

For example:

```
topaz 1> printit
AllUsers disallowUsedPasswords: true .
System commitTransaction
%
```

Disallow a Specific Number of Previous Passwords

Privileges required: OtherPassword and write authorization to DataCuratorObjectSecurityPolicy.

To disallow a fixed number of previously-used password, but allow earlier passwords, use the following:

`AllUsers numberOfDisallowedPasswords:` *anInteger*

anInteger must be a number between 0 and 65535; 0 means that the user may not reuse any previously-used passwords. The limit on the number of disallowed passwords has no effect if disallowUsedPasswords is false.

For example,

```
topaz 1> printit
AllUsers disallowUsedPasswords: true.
AllUsers numberOfDisallowedPasswords: 10.
System commitTransaction
%
```

Clearing a User's Disallowed Old Passwords

Privileges required: OtherPassword.

You can clear the set of old passwords so that they can be reused by sending the message `clearOldPasswords` to that user's UserProfile. As mentioned above, this set is maintained for each user when the AllUsers instance variable disallowUsedPasswords is set to true. The following example clears the remembered passwords for Mary's account:

```
topaz 1> printit
(AllUsers userWithId: 'Mary') clearOldPasswords.
System commitTransaction
%
```

Password Aging – Require Periodic Password Changes

You can configure your system so users must change their password periodically. This can be configured at the repository-wide level, or for individual users. Note that if you are not using GemStone login authorization, password aging does not apply.

Privileges required: OtherPassword and write authorization to DataCuratorObjectSecurityPolicy. The special GemStone accounts are not disabled by password aging.

Repository-Wide Password Aging

You can require users to change their password periodically by setting up a password age limit. This can be set for all users in the repository, and can be overridden for specific individual users.

To set a password page limit for all users in the repository, use the method `UserProfileSet>>passwordAgeLimit:`*numberOfHours*.

For example, to set the limit to 120 days:

```
topaz 1> printit
AllUsers passwordAgeLimit: 120 * 24 .
System commitTransaction
%
```

The passwordAgeLimit is added to the time the password was last changed to determine when the password will expire. A setting of 0 (the default) disables password aging.

Each time this method is invoked, the action is recorded in the Stone's log.

If a user does not change the account's password within the specified interval, the account is disabled, and attempts to log in result in an error.

DataCurator or another user with the OtherPassword privilege can reactivate the disabled account by giving it a new password; see page 206 for details.

Password Age Limits for Individual Users

To override the repository-wide setting for password aging, you can set a password age limit for a specific user, by using the method `UserProfile>>passwordAgeLimit:`*numberOfHours*.

For example, to set the limit for a Mary to 5 days:

```
topaz 1> printit
(AllUsers userWithId: 'Mary') passwordAgeLimit: 5 * 24.
System commitTransaction
%
```

If repository-wide password aging is enabled, you can also override this for individual users, such as managers or administrators, either by setting the passwordAgeLimit for that UserProfile to 0, or by executing **`setPasswordNeverExpires:`** with a true argument.

For example, to prevent the password used by batch jobs from expiring, execute:

```
topaz 1> printit
(AllUsers userWithId: 'BatchUser')
   setPasswordNeverExpires: true.
System commitTransaction
%
```

Repository-Wide Password Expiration Warning

Privileges required: OtherPassword and write authorization to DataCuratorObjectSecurityPolicy. This does not apply to system users, and has no effect if password aging is not enabled.

You can provide an automatic warning to users repository-wide whose password is about to expire by sending the message `UserProfileSet>>passwordAgeWarning:`*numberOfHours*. For example, to warn users who log in within five days of the time their password will expire, do this:

```
topaz 1> printit
AllUsers passwordAgeWarning: 5 * 24.
System commitTransaction
%
```

Logins within *numberOfHours* prior to expiration cause a SecurityError to be signaled.

Per-User Password Expiration Warning

You can provide a similar automatic warning to specific user using `UserProfile>>passwordAgeWarning:`*numberOfHours*.

For example, to warn Mary within three days of the time her password will expire, do this:

```
topaz 1> printit
(AllUsers userWithId: 'Mary') passwordAgeWarning: 3 * 24.
System commitTransaction
%
```

Finding Accounts with Password About to Expire

Privileges required: OtherPassword.

You can find out which accounts have a password within the warning period set by `passwordAgeWarning:`. To do this, send the message `findProfilesWithAgingPassword` to AllUsers. For example:

```
topaz 1> printit
AllUsers findProfilesWithAgingPassword
  collect: [ :u | u userId] .
%
an OrderedCollection
#1 qa1
#2 qa2
#3 qa3
```

Finding Out When a Password Was Changed

Privileges required: OtherPassword.

You can find out the last time the password was changed for a particular userId by sending the message `lastPasswordChange` to that account's UserProfile. This example converts the DateTime returned to a particular pattern based on MM/DD/YY:

```
topaz 1> printit
(AllUsers userWithId: 'qa2') lastPasswordChange US12HrFormat
%
03/24/11 11:28 AM
```

Account Aging – Disable Inactive Accounts

You can configure your system so users must log in periodically, by disabling accounts for which there has been no login for a specified length of time. This can be configured at the repository-wide level, or for individual users.

Privileges required: OtherPassword and write authorization to DataCuratorObjectSecurityPolicy. The special GemStone accounts are not disabled by stale account aging.

Repository-Wide Stale Account Aging

To do this, send the message `staleAccountAgeLimit:` *numberOfHours* to AllUsers. This example disables accounts when they have not logged in for 30 days:

```
topaz 1> printit
AllUsers staleAccountAgeLimit: 30 * 24.
System commitTransaction
%
```

Each time this method is invoked, the action is recorded in the Stone log.

A setting of 0 (the default) disables account aging.

DataCurator or another user with the OtherPassword privilege can reactivate the disabled account by giving it a new password; see page 206 for details.

Per-User Stale Account Aging

You can override the repository-wide setting for account aging for a specific user using `UserProfile>>staleAccountAgeLimit:` *numberOfHours*.

For example, for the 'Auditor' account who may log in less frequently, you can set up a 180-day stale account age limit. This will override a repository-wide 30 day setting.

```
topaz 1> printit
(AllUsers userWithId: 'Auditor')
   staleAccountAgeLimit: 180 * 24.
System commitTransaction
%
```

Finding Out When an Account Last Logged In

Privileges required: OtherPassword.

If at least one age limit applies to an account, you find out when that account last logged in by sending the message `lastLoginTime` to that account's UserProfile. For example:

```
topaz 1> printit
(AllUsers userWithId: 'Mary') lastLoginTime US12HrFormat
%
03/24/11 01:40 PM
```

The time of the last login is maintained only if loginsAllowedBeforeExpiration is set in that UserProfile or if at least one of these instance variables is set in AllUsers

or the specific UserProfile: passwordAgeLimit, passwordAgeWarning, or staleAccountAgeLimit. If none of these features are enabled, the lastLoginTime may be nil, the time of the account creation, or a time representing a login during an earlier period when one of these features was enabled. This is also true if a feature that enables lastLoginTime recording has been enabled on more recently than the last login of the user.

The data curator may explicitly set the time of the last login, using the method `UserProfile >> lastLoginTime:`.

Enabling Account Aging and lastLoginTime

Privileges required: OtherPassword.

The time of the last login is recorded for a UserProfile only if loginsAllowedBeforeExpiration is set in that UserProfile or if at least one of these instance variables is set in AllUsers or the specific UserProfile: passwordAgeLimit, passwordAgeWarning, or staleAccountAgeLimit. This is to avoid the commit during login, which is required to record the lastLoginTime.

As a result, you should use caution in enabling account aging on existing repositories. Enabling account aging may result in user accounts being disabled.

To avoid this, when you enable account aging, you can set the lastLoginTime to the current date, or to nil, for all affected UserProfiles. A nil setting disables account aging checks, allowing the aging period to being with the next login.

```
topaz 1> printit
AllUsers passwordAgeLimit: 120 * 24.
AllUsers do: [:aUserProfile |
   aUserProfile lastLoginTime: DateTime now.
   ].
System commitTransaction.
%
```

Limit Logins Until Password Is Changed

Privileges required: OtherPassword.

When you assign a password to an account, you can make the password temporary by limiting the number of times it can be used. This limitation applies only to a specific account, that is, to the UserProfile that is the receiver of the message. It is intended for use with a new or reactivated account as a means of

ensuring that the user changes the password. For example, the following limits the account "qa2" to two more logins under the current password:

```
topaz 1> printit
(AllUsers userWithId: 'Mary')
   loginsAllowedBeforeExpiration: 2.
System commitTransaction
%
```

A setting of 0 (the default) disables this feature.

The limit remains in effect until the user changes the password (see pages 197). Once the password is changed, the limit for that account is set to 0. The password will not expire again unless a new limit is set by repeating `loginsAllowedBeforeExpiration:`.

If the limit is exceeded before the password is changed, the system disables the account. DataCurator or another user with the OtherPassword privilege can reactivate the disabled account by giving it a new password; see page 206 for details.

The special user accounts are not disabled by this mechanism.

Limit Concurrent Sessions by a Particular UserId

Privileges required: OtherPassword.

You can limit the number of concurrent sessions logged in under a particular userId by sending the message `activeUserIdLimit:` *aPositiveInteger* to the UserProfile for that account.

For example, the following limits the userId "qa2" to four concurrent sessions:

```
topaz 1> printit
(AllUsers userWithId: 'qa2') activeUserIdLimit: 4.
System commitTransaction
%
```

A setting of 0 (the default) disables this feature.

If a user attempts to log in when the maximum number of sessions for that userId are already logged in, the login is denied and a SecurityError is signalled.

Limit Login Failures

Record Login Failures

The Stone repository monitor keeps track of login failures (incorrect passwords) for each account and can write that information to the Stone's log.

By default, messages are logged when the same account fails login attempts 10 or more times within ten minutes. You can change the default limits by setting the STN_LOG_LOGIN_FAILURE_LIMIT and STN_LOG_LOGIN_FAILURE_TIME_LIMIT configuration options (see page 383).

The log message gives the following information:

```
---Fri 13 May 2011 09:39:40 PDT ---
    GemStone user Mary has failed on 10 attempt(s)
       to log in within 1 minute(s).
    The last attempt was from user account writer1 on host
       name docs.
```

Disabling Further Login Attempts

If login failures continue, the Stone repository monitor can disable the account. By default, the account is disabled when the number of failures exceeds 15 within 15 minutes. You can change the default limits by setting the STN_DISABLE_LOGIN_FAILURE_LIMIT and STN_DISABLE_LOGIN_FAILURE_TIME_LIMIT configuration options (page 377).

Subsequent attempts to login as that account result in the following error message:

```
Login failed:  the GemStone userId/password combination is
invalid or expired.
```

The special user accounts are not disabled by login failures.

DataCurator or another user with the OtherPassword privilege can reactivate the disabled account by giving it a new password; see page 206 for details.

Chapter 7

Managing Repository Space

The *repository* is the logical unit that represents the universe of shared objects that are stored within a GemStone/S 64 Bit system. Within GemStone Smalltalk, the repository is the single instance of Class Repository, with the name `SystemRepository`.

The logical repository maps to one or more physical *extent* files in the file system or to data on one or more raw disk partitions. Chapter 1 explains how this mapping is done through GemStone configuration options. Initially, the repository is contained in a single file, `$GEMSTONE/data/extent0.dbf`.

Whenever GemStone performs a *checkpoint,* it makes sure that transactions committed before the checkpoint have been written to the repository extents. The STN_CHECKPOINT_INTERVAL configuration option (page 376) sets the maximum time between checkpoints. (The default is five minutes, but various factors may cause a checkpoint to occur sooner.) The checkpoint limits the amount of time that is needed to recover from a system crash by guaranteeing that the data for the transaction is written to the extent and not just to the transaction log. For information, see "To Control Checkpoint Frequency" on page 67.

This chapter explains how the repository grows, and tells you how to perform a number of administrative tasks related to the repository:

- How to determine the amount of free space in the repository (page 229)
- How to create more space by adding an extent while the repository is in use (page 231)
- How to remove an extent (page 233)
- How to reallocate objects among extents (page 234)
- How to recover from an error caused by a full disk (page 240)

7.1 Repository Growth

The repository begins in the compact form of `$GEMSTONE/data/extent0.dbf`. As repository activity progresses, the extent file expands for a variety of reasons, in increments of 1 MB or more. Not only does your new application data require space, but space is required for internal structures that organize and manage the objects. Sessions, and their transactional views of the repository, also require space.

Garbage objects—objects that are no longer referenced, or the older versions of objects, also use space in the repository until they are garbage collected. To manage the size of the extents, you need to regularly perform garbage collection on your GemStone repository. The frequency can vary from monthly to daily, depending on the amount of activity in the repository. Without garbage collection, the repository will continue to grow and be filled with wasted space. See Chapter 11, "Managing Growth," for a discussion of garbage collection in GemStone.

7.2 How To Check Free Space

Use the methods `Repository>>fileSize` and `Repository>>freeSpace` to obtain reports about the logical repository as a whole.

The result of the message `fileSize` is the total size of the repository in bytes, including all extent files. If the repository consists of a single extent file, it is ordinarily the same result as you would obtain by using the operating system command **ls -l** *extentName*. For example:

```
topaz 1> printit
SystemRepository fileSize
%
52428800
```

The result of `freeSpace` tells how much space (in bytes) is available for allocation within the repository at its current size. Free space is equal to the sum (for all extents in the repository) of the number of free pages in each extent multiplied by the page size. This space does not include fragments on partially filled data pages.

```
topaz 1> printit
SystemRepository freeSpace
%
26411008
```

Depending on the configuration options selected and the available disk space, the Stone repository monitor may be able to create additional free space by enlarging the repository.

If your configuration has more than one extent, use `Repository>>fileSizeReport` to generate statistics about each individual extent and also totals for the entire repository. (The heading "Extent #1" identifies the primary extent regardless of its file name, which initially is `extent0.dbf`.)

For example:

```
topaz 1> printit
SystemRepository fileSizeReport
%
Extent #1
-----------
   Filename = /users/extents/extent0.dbf

   File size =       22.00 Megabytes
   Space available = 5.56 Megabytes

Extent #2
-----------
   Filename = /users/extents/extent1.dbf

   File size =       6.00 Megabytes
   Space available = 0.12 Megabytes

Totals
------
   Repository size = 28.00 Megabytes
   Free Space =      5.69 Megabytes
```

The amount of free space in the repository can also be determined from the cache statistic FreePages (see page 487). To obtain the free space, multiply FreePages by the page size, 16384.

7.3 How To Add Extents

GemStone provides two ways to add extents:

- You can add new extents at startup by editing your GemStone configuration file and adding extent names and sizes to the DBF_EXTENT_NAMES and DBF_EXTENT_SIZES configuration options (page 360). Append the new values to the existing entries, just before the semicolon (;) delimiter. The new extents will be created the next time the Stone starts up.
- You can add extents while the Stone is running by invoking the Smalltalk methods described in this section. These methods are especially useful in avoiding or resolving low disk space conditions because the change takes effect immediately.

To Add an Extent While the Stone is Running

To prevent the repository from becoming full, you can dynamically add another extent specification (a file or a raw partition) to the configuration file for the Stone. This section describes the Smalltalk methods that allow you to do this.

For general information about multiple extents, see "To Configure the Repository Extents" on page 43.

Possible Effects on Other Sessions

When a new extent is dynamically added to the logical repository through Smalltalk, sessions that are currently logged in must have access to the new extent. The possibility exists that an online session may terminate because it cannot open a new extent. Reasons for this condition could range from the inability to start a remote page server process to file permission problems.

CAUTION

The operating system creates the new extents with the ownership and permissions of the Stone repository monitor process. If these permissions are not the same as for other extents, you should use operating system commands to modify them as soon as possible. Such changes can be made without stopping the Stone.

A session's view of which files make up the logical repository is updated whenever one of the following events occurs:

- Users commit or abort the session.
- The Stone repository monitor hands out disk resources to the session.

An explicit **commit** or **abort** may succeed but then cause the session to be terminated because of the inability to mount new extents immediately after the **commit** or **abort** operation.

Repository>>createExtent:

Privileges required: FileControl.

The Smalltalk method `createExtent:`*extentFilename c*reates a new repository extent with the given extent file name (specified as a String). The new extent has no maximum size. The extent must be located on the machine running the Stone process. For example:

```
topaz 1> printit
SystemRepository createExtent: '$GEMSTONE/data/extent2.dbf'
%
```

You can execute this method when other users are logged in.

The Stone creates the new extent file, and it also appends the augmented extent list to your configuration file:

```
# DBF_EXTENT_NAMES written by Stone (user Bob) on 4/12/11
12:56:18 PDT
DBF_EXTENT_NAMES = "$GEMSTONE/data/extent0.dbf",
"$GEMSTONE/data/extent1.dbf",
"!TCP@mozart#dbf!/users/gemstone/data/extent2.dbf;"
```

If the given file already exists, the method returns an error and the specified extent is not added to the repository.

Creating an extent with this method bypasses any setting you may have specified for the DBF_PRE_GROW option at system startup (page 361). Because extents created with this method have no maximum size, they cannot be pre-grown. If the repository is using weighted allocation, the new extent will be given a weight equal to the average weight of all other extents. (For a discussion of weighted allocation, see page 43.)

If this method is run from a session on a host remote from the Stone, *extentFilename* must include a Network Resource String (NRS) specifying the Stone host. The syntax is shown above in the excerpt from the augmented configuration file. For information about NRS syntax, see Appendix C.

Repository>>createExtent:withMaxSize:

Privileges required: FileControl.

The Smalltalk method `createExtent:`*extentFilename* `withMaxSize:`*aSmallInteger* creates a new repository extent with the specified *extentFilename* and sets the maximum size of that extent to the specified size. You can execute this method when other users are logged in.

The size must be a non-zero positive integer representing the maximum physical size of the file in MB.

If the specified extent file already exists, this method returns an error and the extent is not added to the logical repository.

If the configuration file option DBF_PRE_GROW is set to True (page 361), this method will cause the newly created extent to be pre-grown to the given size. If the pre-grow operation fails, then this method will return an error and the new extent will not be added to the logical repository.

Repository>>createExtent:withMaxSize:startNewReclaimGem:

Privileges required: FileControl.

Similar to the previous methods, `createExtent:`*extentFilename* `withMaxSize:`*aSmallInteg*er `startNewReclaimGem:` *aBoolean* creates a new repository extent with the specified *extentFilename* and sets the maximum size of that extent to the specified size. In addition, if *aBoolean* is true, a new reclaim GcGem is started to reclaim pages on this extent.

Since ReclaimGcGems are assigned to a specific range of extents, adding a new extent without using this method may mean that there is no shadow or dead objects on pages in the new extent are reclaimed, so garbage may build up in this extent.

7.4 How To Remove an Extent

The only way to remove an extent file is by first performing a Smalltalk full backup and restore to move the contents of that extent to other extents.

Privileges required: FileControl.

Reducing the number of existing extents requires special steps to ensure data integrity. If you must remove an extent file, follow this procedure:

Step 1. Back up your repository using the Smalltalk full backup procedure described on page 264.

You cannot use an online or offline extent backup to remove or shrink extents.

Step 2. Shut down the Stone repository monitor.

Step 3. Modify the DBF_EXTENT_NAMES configuration parameter (page 360) to show the new extent structure.

Step 4. Restore the repository from your Smalltalk full backup. Follow the GemStone restore procedure described on page 269.

7.5 How To Reallocate Existing Objects Among Extents

If you want to reallocate existing objects among two or more extents, the procedure depends in part on whether you are also changing the number of extents. Because changes to DBF_ALLOCATION_MODE directly affect only the subsequent allocation of pages for new or modified objects, additional steps are necessary.

To Reallocate Objects Among a Different Number of Extents

If you are increasing or decreasing the number of extents and want to change allocation of existing objects as part of that operation, perform a Smalltalk full backup, then restore the backup after setting appropriate weights in the DBF_ALLOCATION_MODE configuration option (page 360).

For example, suppose your existing repository contains 800 MB and you want to divide them about equally between the existing extent and a new one. To populate each extent with about 400 MB, follow this procedure:

Step 1. Back up your repository, using the Smalltalk full backup procedure described on page 264.

Step 2. Shut down the Stone repository monitor.

Step 3. Modify the DBF_EXTENT_NAMES configuration parameter to show the new extent structure.

```
DBF_EXTENT_NAMES = $GEMSTONE/data/extent0.dbf,
$GEMSTONE/data/extent1.dbf;
```

Step 4. Edit the DBF_ALLOCATION_MODE configuration option to reflect the intended distribution of pages (see "Allocating Data to Multiple Extents" on page 47). For example:

```
DBF_ALLOCATION_MODE = 10, 10;
```

Step 5. Restore the repository from your Smalltalk full backup, using the procedure beginning on page 269.

If objects in the repository were explicitly clustered using instances of ClusterBucket that explicitly specified the first extent, those objects may tend to migrate back to that extent over time. You can prevent such migration by placing size limits on the existing extent, or by explicitly reclustering those objects in the new extent using a ClusterBucket that specifies either an extentId of nil or the extentId of the new extent. For information about clustering, refer to the *GemStone/S 64 Bit Programming Guide*.

To Reallocate Objects Among the Same Number of Extents

Changes to DBF_ALLOCATION_MODE (page 360) directly affect only the subsequent allocation of pages for new or modified objects. When you restore into a repository with the same number of extents, the distribution of the original repository will be used in the restored repository, regardless of the DBF_ALLOCATION_MODE.

To change the allocation of existing objects, you can either restore into a repository with a different number of extents (as discussed on page 234) or you can specify a maximum size on the extent files, to force objects to be distributed as you want them.

For example, suppose your existing repository has three extents, and that you are running in sequential allocation mode. The first extent has 600 MB, while the second and third extents are 1 MB each (the minimum size). You now want to redistribute the objects so they are spread evenly over all three extents. You cannot simply change the DBF_ALLOCATION_MODE, and restore a backup into three extents; existing objects would be distributed according to the original allocation mode, that is, entirely in the first extent. Only new objects created after the restore would be created evenly over the three extents.

To populate the three extents evenly, you can follow this procedure:

Step 1. Back up your repository, using the Smalltalk full backup procedure described on page 264.

Step 2. Shut down the Stone repository monitor.

Step 3. Edit the DBF_EXTENT_SIZES configuration option (page 361) to limit the size of the first extent temporarily to 200 MB. For example:

```
DBF_EXTENT_SIZES = 200, , ;
```

Step 4. Edit the DBF_ALLOCATION_MODE configuration option (page 360) to reflect the intended distribution of pages (see "Allocating Data to Multiple Extents" on page 47). This setting determines the distribution of new or modified objects. For example:

```
DBF_ALLOCATION_MODE = 10, 10, 10;
```

Step 5. Restore the repository from your Smalltalk full backup, using the procedure beginning on page 269.

Step 6. If you want the first extent to grow beyond the temporary limit you set in Step 3, stop the Stone after you restore the repository. Edit the configuration file again, either specifying a higher limit or no limit. For example:

```
DBF_EXTENT_SIZES = , ,  ;
```

If objects in the repository were explicitly clustered using instances of ClusterBucket that explicitly specified the first extent, those objects may tend to migrate back to that extent over time. You can prevent such migration by maintaining the size limit set in Step 3, or by explicitly reclustering those objects in the new extent using a ClusterBucket that specifies either an extentId of nil or the extentId of the new extent. For information, refer to the *GemStone/S 64 Bit Programming Guide*.

7.6 How To Shrink the Repository

Privileges required: SystemControl, GarbageCollection, and FileControl.

To shrink the repository to its minimum size, make a Smalltalk full backup. Then take the repository offline and restore the backup into a copy of the GemStone distribution repository. Use the following procedure, which compacts the repository into the minimum set of consecutive pages.

Step 1. Mark your repository for garbage collection. For example:

```
topaz 1> printit
SystemRepository markForCollection
%
```

For further information about this method, see "MarkForCollection" on page 313.

Step 2. Wait for GemStone to complete the garbage collection and reclaim the space. The time required depends on several factors: the size of the repository, the number of Reclaim GcGems currently running, and (in multi-user mode) the status of other sessions. For details, see "Running Reclaim GcGems" on page 333.

If other users are logged in, space will not be reclaimed until all sessions have committed or aborted any transactions concurrent with the `markForCollection` process.

Step 3. Make a Smalltalk full backup of your repository by sending it the message `fullBackupTo:`*fileOrDevice* or `fullBackupTo:`*fileOrDevice* `MBytes:`*byteLimit*.

For example:

```
topaz 1> printit
SystemRepository fullBackupTo: '/users/bk/april20'.
%
```

This example writes the backup to a single disk file. If you need to write multiple files, see "To Create a Backup in Multiple Files" on page 267.

Step 4. Take the repository offline:

```
topaz 1> printit
System shutDown
%
```

Step 5. Remove the existing repository extents. Obtain a copy of the distribution repository as the first (primary) extent by using the **copydbf** command. For example, assuming that all of your GemStone extents are in `$GEMSTONE/data`:

```
% cd $GEMSTONE/data
% rm extentNames
% copydbf $GEMSTONE/bin/extent0.dbf primaryExtentName
```

Use **chmod** to set the extent permission to what you ordinarily use for your repository.

Step 6. To put the repository back online, issue the **startstone** command:

```
% startstone gemStoneName
```

If you do not specify *gemStoneName*, **startstone** defaults to `gs64stone`.

Step 7. Log in to linked Topaz again.

NOTE

To perform the remaining parts of this procedure, you must be the only user logged in to GemStone. Logins will be disabled when you start the next step.

Step 8. Restore the repository by using the method `Repository>>restoreFromBackup:`*fileOrDevice*, using the same file or device as in Step 3. Because it is being restored into a copy of the initial repository, the restored repository will be compressed to the minimum space. This example restores the backup from a single disk file:

```
topaz 1> printit
SystemRepository restoreFromBackup: '/users/bk/april20'
%
```

If you need to restore multiple files, use the method `Repository>>restoreFromBackups:`*fileOrDeviceArray* instead:

```
topaz 1> printit
SystemRepository restoreFromBackups:
#( '%/backups/April_20.1'
   '%/backups/April_20.2' )
%
```

(For more information, see "To Restore Multiple-File Backups" on page 278.)

GemStone reads the backup(s) and rebuilds the repository in a "shadow" object space that is invisible to users at this time. If the restore succeeds, GemStone commits the restore and returns a summary in the form of a nonfatal error message like the following:

```
Restore from full backup completed with 51569 objects
restored and 0 corrupt objects not restored.
```

Each restore operation, on completion, terminates its GemStone session. You will need to log in again before performing the next restore operation.

Step 9. Between the time the full backup was started (Step 3) and the time the repository was shut down (Step 4), there may have been transactions on your

repository. To ensure that no work is lost, restore from transaction logs and commit the restore. For example:

```
topaz 1> printit
SystemRepository restoreFromCurrentLogs
%
Restore from transaction log(s) succeeded.

topaz> login
<details omitted>
successful login

topaz 1> printit
SystemRepository commitRestore
%
Restore from transaction log(s) succeeded. commitRestore
succeeded
```

7.7 How To Check Page Fragmentation

Space within the repository is managed in pages having a fixed size of 16 KB. It is possible for these pages to become *fragmented*—that is, only partially filled with objects. GemStone automatically schedules reclamation of pages with greater than 10% free space as part of its garbage collection activity.

You can inquire about the amount of fragmentation in the repository by executing the following expression.

SystemRepository pagesWithPercentFree:*aPercentage*

(Typical values of *aPercentage* range from 10 to 25.)

This method returns an array containing the following statistics:

- The total number of data pages processed
- The sum (in bytes) of free space in all pages
- The page size (in bytes)
- The number of data pages having at least the specified percentage of free space
- The number of data pages having at least the specified percentage of free space, that contain only a single object
- The total number of pages in the repository that contain only a single object

7.8 How To Recover from Disk-Full Conditions

The Stone repository monitor has two critical needs for disk space. It must be able to:

- Append to the transaction log as sessions commit changes.
- Expand the repository as necessary to allocate free pages to current sessions or to sessions logging in.

Whenever the Stone cannot log transactions or cannot find sufficient free space in the repository, it issues an error message to any session logged in as DataCurator or SystemUser. If users report that GemStone appears to be hung or that they get a disk-full error while logging in, you should check one of these administrative logins for such a message. The message is also written to the Stone's log file.

The following topics explain the Stone's actions in greater detail and describe steps you can take to provide sufficient space.

For details on how tranlog full conditions are handled, see "How To Recover from Tranlog-Full Conditions" on page 255

Repository Full

The Stone takes a number of actions to prevent the repository from becoming completely full. If the free space remaining in the repository falls below the level set by the STN_FREE_SPACE_THRESHOLD configuration option (page 380) and the Stone cannot allocate more space in any extent, it takes the following actions to prevent a system crash:

1. It becomes more aggressive about disposing of commit records so that garbage collection can proceed. (If the Stone is very busy, a backlog of commit records can accumulate.)
2. It starts a checkpoint if there isn't one in progress and reduces the checkpoint interval to three minutes until the condition clears. (This checkpoint may free pages that have been reclaimed.)
3. It writes a message to the Stone log to indicate the condition.
4. It prevents new logins except for DataCurator and SystemUser accounts. It issues a disk-full error to other sessions attempting to log in.
5. It signals the Exception RepositoryError to any sessions logged in (or logging in) as DataCurator or SystemUser so that they know disk space is becoming critical.

6. It signals Gem session processes to return all except five free pages per extent. It responds to requests for additional pages by allocating only five pages at a time.
7. If the free space available drops below 400 KB (50 pages), the Stone stops responding to page requests from sessions that are not logged in as DataCurator or SystemUser. This action prevents users from acquiring all of the available space, which would cause the system to crash. Gem session processes appear to "hang" while they are waiting for pages. The unhonored page requests are granted when the free space goes back above the threshold.
8. If the previous steps do not solve the problem within the time specified by the STN_DISKFULL_TERMINATION_INTERVAL (page 378), then the Stone begins to terminate sessions holding on to the oldest commit record *even if the session is in a transaction*. This action applies to any user session, including logins as SystemUser and DataCurator. Allowing the Stone to dispose of the commit record allows additional garbage collection.

   NOTE

   You can configure the Stone to never terminate sessions by setting STN_DISKFULL_TERMINATION_INTERVAL *to 0; however, doing so increases the risk of GemStone shutting down because of a lack of free space in the repository.*

9. When the condition clears, another message is written to the Stone log and operation returns to normal.

If you see a message like the following while logged in as DataCurator or SystemUser or in the Stone log, disk space is becoming critical:

```
The repository is currently running below the
freeSpaceThreshold.
```

When the system must dynamically expand the repository, it checks one extent at a time, in the order dictated by the allocation strategy, to see if that extent can be expanded to create more space. When no extents can be extended and the free space is below STN_FREE_SPACE_THRESHOLD, the Stone takes the actions described above.

Failure to expand an extent has two possible causes: either the disk containing the extent is full, or the extent has reached its maximum size as set by the DBF_EXTENT_SIZES configuration option.

There are a number of things you can do to create more space in an existing extent, or you can create a new extent. Each of these actions may create sufficient additional space for immediate needs:

- Warn the current users about the problem, and have them log out until enough space is made available.
- Remove any non-essential files to create enough space for expanding the repository.
- Create a new extent through Smalltalk with `Repository>>createExtent:`*extentFileName* or an equivalent method. If the Stone has stopped, you can create a new extent by editing the parameters in the configuration file before restarting it. See "How To Add Extents" on page 231.

Chapter

8 Managing Transaction Logs

8.1 Overview

A transaction log contains the information necessary to redo transactions to the repository that have been committed by GemStone sessions since the last checkpoint or orderly shutdown. This log is used to recover from crashes such as those caused by a power failure, an operating system failure, or the killing of GemStone monitor processes.

If you need to restore the repository from a backup, transaction logs written in the optional full-logging mode can be used to recreate all transactions committed since the most recent backup was written.

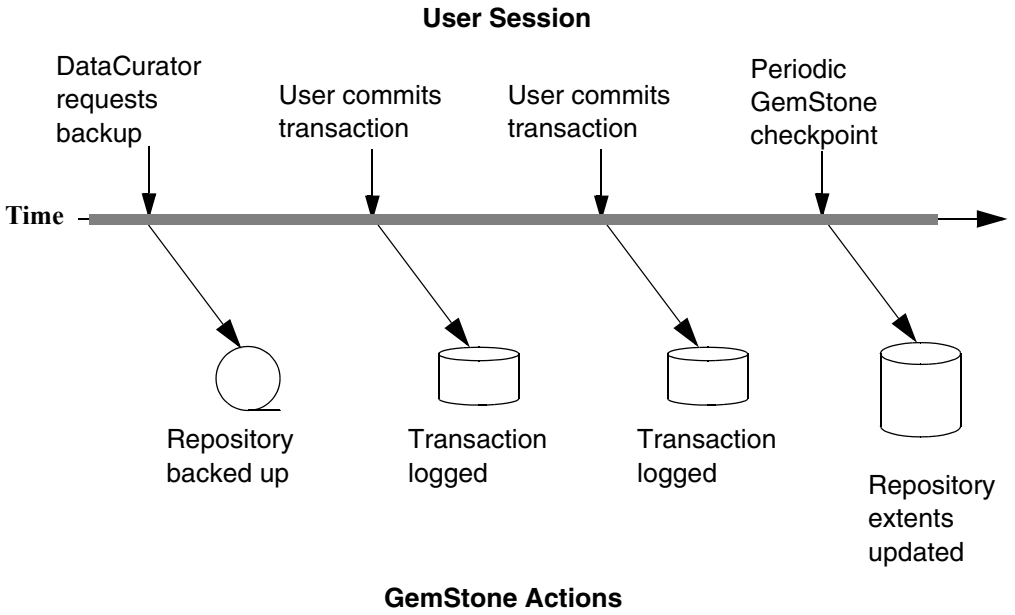
The transaction log is implemented as a sequence of files having names of the form `tranlog0.dbf` ... `tranlogNNN.dbf`. The numeric `fileId` starts at 0 when the Stone starts with a copy of the initial repository extent (`$GEMSTONE/bin/extent0.dbf`), and there are no existing transaction logs with that tranlog id in any transaction log directories. If the Stone starts on an existing repository without any logs present, the `fileId` will be one greater than when the repository was last shut down cleanly. You can control the filename prefix by setting the STN_TRAN_LOG_PREFIX configuration option (page 391).

These logs are written to a list of directories or raw partitions specified by the STN_TRAN_LOG_DIRECTORIES configuration option (page 391), which is treated

as a circular list. Each log is limited to the size set for that directory or raw partition by STN_TRAN_LOG_SIZES (page 392). When one log is full, logging switches to the next directory or raw partition. (What happens when logs have been created in all directories is discussed in Table 8.1 on page 246.) Collectively, the transaction log files logically form an extremely large sequential file with a maximum size of 4×10^6 GB.

Between checkpoints, GemStone writes each committed transaction to a transaction log (Figure 8.1).

Figure 8.1 Normal Operation



Use ordinary UNIX utilities to backup the transaction logs in the file system. To backup a transaction log that is on a raw disk partition, use **copydbf** to copy it to a file system. You'll also need to use **removedbf** to clear the partition for reuse.

Logging Modes

GemStone provides two modes of transaction logging, selected by setting the STN_TRAN_FULL_LOGGING configuration option:

- To provide real-time incremental backup of the repository, set STN_TRAN_FULL_LOGGING to True. All transactions are logged regardless of their size. This mode is recommended for deployed GemStone systems.
- To allow a simple operation to run unattended for an extended period, set STN_TRAN_FULL_LOGGING to False (the initial setting). This mode, known as *partial* logging, provides limited backup that ordinarily permits automatic recovery from system crashes that do not corrupt the repository.

Table 8.1 compares full and partial transaction logging.

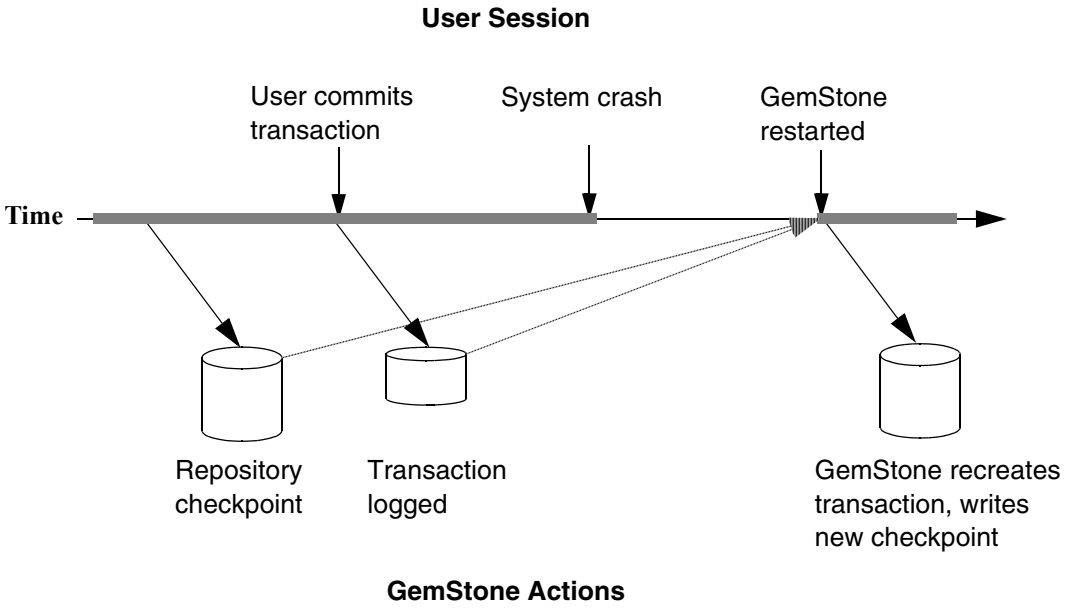
Table 8.1 Comparison of Full and Partial Transaction Logging

Characteristic	STN_TRAN_FULL_LOGGING =TRUE (Full Logging)	STN_TRAN_FULL_LOGGING =FALSE (Partial Logging)
Type of transaction logged	All transactions	Only those transactions smaller than STN_TRAN_LOG_LIMIT; successful commits of larger transactions cause an immediate checkpoint
Recovery from system crash (extents are OK)	Yes, automatic during restart using checkpoint and log	Yes, automatic during restart using checkpoint and log
Replay of transactions since last backup (as after media failure)	Yes—can carry forward GemStone backup by recreating subsequently committed transactions	No—cannot replay transactions since the backup
Action when current log is full	Logging moves to the next directory or to the head of the list. If it is a file system directory, the Stone opens a new log file there; existing transaction logs are retained. If it is a raw partition, a new log can be opened only if the previous one has been archived and removed. The maximum number of file system logs online at one time depends on disk space. The maximum number of raw partition logs depends on the number of partitions listed in STN_TRAN_LOG_DIRECTORIES.	Logging moves to the next directory or to the head of the list. The Stone removes the existing transaction log before opening a new one. The maximum number of logs on line at one time depends on the number of directories or raw partitions in the list.
Action when log space becomes full	The Stone pauses execution if it cannot find space in any of the specified directories or raw partitions.	The Stone deletes log files from the circular list of directories and keeps running.
Administrative task	Monitor log space; archive log files and delete them as necessary	None

Recovering from an Unexpected Shutdown

In the event of a system crash, GemStone can recover by automatically replaying transactions recorded in the transaction log, from the latest checkpoint before the crash to the end of the log at the moment of the crash (Figure 8.2). This allows you to restart from where you were at the time of the crash.

Figure 8.2 System Crash



Restoring Transactions to a Backup

An important use of transaction logs is to restore transactions that were committed between the last full backup and a system failure. If you experience system failure and your current extents are no longer usable, you can still recover all data, provided that the repository already is in full transaction logging mode and that the backup was made in that mode.

You can use the transaction logs to restore transactions committed since the last GemStone online extent backup or a Smalltalk full backup. The following steps show what you must do to prepare:

Step 1. Change the STN_TRAN_FULL_LOGGING configuration option to True.

Step 2. Restart GemStone.

Step 3. Make a GemStone online extent backup (page 259), offline extent backup (page 286), or a Smalltalk full backup (page 264).

How the Logs Are Used

You can only use transaction logs to restore your system from backup if you are in full logging mode.

After creating a backup, you need to retain all the transaction logs that are created during and after the backup. To determine the oldest transaction log that will be required, use the `copydbf` utility to query the backup file. For example:

```
unix> copydbf -i backup.dat
Source file: backup.dat
   file type: backup  fileId: 0
   byteOrder: Intel (LSB first)  compatibilityLevel: 844
   The previous file last recordId is  -1.
Destination file:  /dev/null
      byteOrder: Intel (LSB first)
   Full backup started from checkpoint at: 05/20/11
17:16:35 PDT.
   Oldest tranlog needed for restore is fileId 4 (
tranlog4.dbf ).
```

For this example, tranlog4.dbf and later must be available. If any of these transaction logs are deleted or lost, you will not be able to recover. These transaction logs may be archived elsewhere, as long as they can be readily be made available if you do need to restore from backup.

If you do need to restore your system, first restore from backup using the using the appropriate procedure:

- To restore from an online extent backup, see page 263.
- To restore from a Smalltalk full backup, see page 269.

Then, replay transactions committed since the backup by reading the transaction logs in the order in which they were generated. The restore procedures in Chapter 9 (listed above) have instructions on how to do this.

NOTE

Restoring a repository from backup resets it - both the GemStone kernel and your application classes and data - to the state it was in at the time the backup was created. Anything that was done after that can be recovered only by replaying transaction logs in order, or by restoring a more recent backup.

8.2 How To Manage Full Logging

When the system is operating with the STN_TRAN_FULL_LOGGING configuration option set to True, you (as system administrator) should monitor the available log space. If the log space defined by STN_TRAN_LOG_DIRECTORIES becomes full, users will be unable to commit transactions to the repository until space is made available.

For transaction logs in file system directories, "full" means that there is no free space in the file systems containing those directories.

For transaction logs in raw partitions, "full" means that all partitions listed already contain a GemStone transaction log or other repository file. After archiving an existing log, you must invoke **removedbf** (page 406) before that partition can be reused.

There are two recovery situations to consider in managing transaction logs under full logging:

- Recovery from a system crash requires logs for all transactions committed since the last *checkpoint*. Because of the way GemStone logs changes involving large objects, parts of these transactions may be in earlier logs. The method `Repository>>oldestLogFileIdForRecovery` returns the fileId of the oldest log that would be needed if a crash were to occur at that point. All logs needed for crash recovery should be kept online.
- Recovery from damaged extents, such as a media failure, requires all transaction logs since the last *backup*, and earlier logs may be needed if lengthy transactions were in progress at the time the backup started. Log files not needed for crash recovery may be archived offline, although restoring them will take longer.

To Archive Logs

Ordinary UNIX tools, such as **tar** and **cp**, can be used to move log files to other locations or to archival media. We recommend that you archive and free one complete log directory at a time, in the order listed in the STN_TRAN_LOG_DIRECTORIES configuration option (page 391).

NOTE

*If you must rename the log files, we recommend that you preserve the digits in the original filename as an aid to restoring the files in sequence should that become necessary. If your transaction logs are in raw disk partitions, **copydbf** adds the fileId when you copy a log to a file system directory.*

Two special commands are provided for working with raw disk partitions:

- The **copydbf** command copies a repository file (extent, transaction log, or full backup) to or from a raw disk partition. If the destination is a directory in the file system, **copydbf** generates a filename that includes the file type and its internal fileId.
- The **removedbf** command writes over a raw partition so that GemStone will no longer think it contains a repository file.

Both the **copydbf** and **removedbf** commands can be used with a remote node even if it is not running NFS (a NetLDI must be running on that node). For further information about **copydbf** and **removedbf**, see the command descriptions in Appendix B.

To determine the current size of a transaction log that is in a raw partition, use the method `Repository>>currentTranlogSizeMB`. This method returns the log size (in MB) as an Integer.

To determine the oldest transaction log that would be needed to recover from the most recent checkpoint, use the method `Repository>>oldestLogFileIdForRecovery`. This method returns the internal fileId, which is part of the filename for transaction logs in the file system. If a session was in a lengthy transaction at the time of a system crash, the oldest log needed during recovery may be one that was written before the last checkpoint occurred.

You can obtain similar information by applying **copydbf -i** to an extent. For example:

```
% copydbf -i extent0.dbf
Source file: extent0.dbf
   file type: extent  fileId: 0
   byteOrder: Sparc (MSB first) compatibilityLevel: 844
   Last checkpoint written at: 04/04/11 15:30:30 PDT.
   Oldest tranlog needed for recovery is fileId 5 (
   tranlog5.dbf ).
   Extent was shutdown cleanly; no recovery needed.
```

To determine the oldest transaction log needed to roll forward from a backup, apply `copydbf -i` to the backup:

```
% copydbf -i backup.dat
Source file: backup.dat
   file type: backup  fileId: 0
   byteOrder: Sparc (MSB first) compatibilityLevel: 844
   The previous file last recordId is  -1.
Destination file:  /dev/null
   byteOrder: Sparc (MSB first)
   Full backup started from checkpoint at: 04/04/11 15:28:37
   PDT.
   Oldest tranlog needed for restore is fileId 5 (
   tranlog5.dbf ).
```

For an example script showing how to archive transaction logs out of raw partitions, see `$GEMSTONE/examples/archivelog.sh`. You will need to edit the script to conform to your own partition names and archive location, and then test it.

To Add a Log at Run Time

Privileges required: FileControl.

You can add a directory or a raw partition for transaction logs to the existing list without shutting down the Stone repository monitor. When you do this, the repository monitor also records the change in its configuration file so that the addition becomes permanent. Send the following message:

`SystemRepository addTransactionLog:` *deviceOrDirectory* `size:` *aSize*

For example:

```
topaz 1> printit
SystemRepository addTransactionLog: '/users/tlogs2' size: 8
%
```

The argument *aSize* sets the maximum log size (in MB) for *deviceOrDirectory*. It will be added to the list in STN_TRAN_LOG_SIZES (page 392).

You can use the method `System class>>stoneConfigurationAt:` to examine the contents of STN_TRAN_LOG_DIRECTORIES at run time. For information, see "How To Access the Server Configuration at Run Time" on page 62. The Repository methods in Table 8.2 return other information that is helpful in managing transaction logs.

Table 8.2 Repository Methods for Information About Transaction Logs

Method	Description
`currentLogDirectoryId`	Returns a positive SmallInteger, which is the one-based offset of the current log file into the list of log directory names.
`currentLogFile`	Returns a String containing the name of the transaction log file to which records currently are being appended.
`currentTranlogSizeMB`	Returns an Integer that is the size (in MB) of the currently active transaction log.
`logOriginTime`	Returns the log origin time of the receiver, the time when a new sequence of log files was started. For details, see the method comment in the image.
`oldestLogFileIdForRecovery`	Returns a positive SmallInteger, which is the internal fileId of the oldest transaction log needed to recover from the most recent checkpoint, if the Stone were to crash as of now.
`restoreStatusOldestFileId`	Returns a SmallInteger, which is the internal fileId of the oldest transaction log needed for the next restore from log operation.

To Force a New Transaction Log

Privileges required: FileControl.

You can force closure of the current log and opening of a new log at almost any time by using the method `Repository>>startNewLog`. The method performs the following sequential actions:

1. Starts a checkpoint.
2. Waits till the checkpoint completes.
3. Starts the new log.
4. Returns a SmallInteger, which is the `fileId` of the new log.

In the following example, the new transaction log file would be `tranlog9.dbf`.

```
topaz 1> printit
SystemRepository startNewLog
%
9
```

If a checkpoint is already in progress when you execute `startNewLog`, the method will fail and return –1 instead. If you're using this method in an application, therefore, you need to accommodate the possibility of such a failure with code such as:

```
| id  |
id := SystemRepository startNewLog.
[ id < 0 ] whileTrue: [
  System sleep: 1.
  id := SystemRepository startNewLog ].
```

To Start Checkpoints

Privileges required: SystemControl.

Class System (category Transaction Control) provides two methods that you can use to start checkpoints manually. These methods do not commit, abort, or otherwise modify the current transaction.

`startCheckpointSync`

Starts a new checkpoint and returns when the new checkpoint has completed. If a checkpoint is already in progress, this method waits until the current checkpoint completes, then starts a new checkpoint. Returns true if a new checkpoint was successfully completed, returns false if a new checkpoint could not be started because transaction logs are full or checkpoints are suspended.

`startCheckpointAsync`

Starts a new checkpoint if a checkpoint is not already in progress and returns immediately, without waiting for the checkpoint to finish. Returns true if a checkpoint is in progress or was started; returns false if a checkpoint could not be started because transaction logs are full or checkpoints are suspended.

To Change to Partial Logging

Once the full transaction logging has been started on a repository, the STN_TRAN_FULL_LOGGING state of True persists regardless of later changes to the configuration file. To terminate full logging, use the following procedure:

Step 1. Perform a Smalltalk full backup using `Repository>>fullBackupTo:`. See "How To Make a Smalltalk Full Backup" on page 264.

Step 2. Edit the configuration file to set the STN_TRAN_FULL_LOGGING option to False.

Step 3. Stop the Stone repository monitor.

Step 4. Replace the first (primary) extent file with a copy of `$GEMSTONE/bin/extent0.dbf`. Delete any other extent files.

Step 5. Restart GemStone.

Step 6. Restore the backup using `Repository>>restoreFromBackup:` or `restoreFromBackups:`. See "How To Restore from a Smalltalk Full Backup" on page 269.

8.3 How To Manage Partial Logging

Partial logging is GemStone's default mode because it provides ease of administration with protection against loss of data from system crashes. The Stone repository monitor treats the log directories as a circular list. If the file in the current directory reaches the limit set by STN_TRAN_LOG_SIZES, the Stone switches to the next directory in the list that does not contain a transaction log. In the process of creating log *n*, the Stone attempts to find and delete log (*n* – size_of_STN_TRAN_LOG_DIRECTORIES); for example, if the new log will be `tranlog7.dbf` and there are three elements in STN_TRAN_LOG_DIRECTORIES, the Stone searches all three in attempting to delete `tranlog4.dbf`.

You should ensure that there always is sufficient disk space for at least two log files (their default size is 10 MB each), so that one can be preserved when the next is opened.

To Change to Full Logging

To change a repository from partial to full logging, simply change the STN_TRAN_FULL_LOGGING setting to True (page 390) and restart the Stone repository monitor.

CAUTION

Be sure to make a new GemStone full backup in full-logging mode so that you will be able to restore from the transaction logs if necessary. Transaction logs cannot be restored to backups that were made in partial-logging mode.

8.4 How To Recover from Tranlog-Full Conditions

Transaction Log Space Full

If the space for transaction logs becomes full, the Stone stops processing commits or other requests that initiate a write to the transaction log. Sessions performing these operations are blocked until the condition is resolved and may appear to the user to be hung. The Stone writes a message like the following in its log:

```
The tranlog directories are full and the stone process is
waiting for an operator to make more space available by
either cleaning up the existing files (copying them to
archive media and deleting them) or by adding a new tranlog
directory.
```

Also, the Exception `RepositoryError` is signaled in any sessions that have enabled receipt of this error by sending `System enableSignalTranlogsFull`, and setting up a handler for this error.

Once enabled, you can disable receipt of this error by sending `System disableSignalTranlogsFull`, and determine the current status by `System signalTranlogsFullStatus`.

If the transaction log space is full, you have the following options:

- You can free space by taking some existing log files offline. Archive them using operating system utilities and then remove them. GemStone can reuse that slot in the circular list of log directories. (To archive and remove a log file from a raw partition, use **copydbf** and then **removedbf**.)
- You can increase the available log space by adding a raw partition or a directory on another disk drive to the STN_TRAN_LOG_DIRECTORIES configuration option (page 391). Add its maximum file size to STN_TRAN_LOG_SIZES. For information on how to make these changes while GemStone is running, see "To Add a Log at Run Time" on page 252.

When transaction log space becomes available, waiting sessions can complete operations that were blocked.

Chapter

9 Making and Restoring Backups

This chapter explains describes how to make backups of your GemStone/S 64 Bit repository and, should it become necessary, how to use the backups to restore the repository.

This chapter includes the following topics:

- How To Make an Online Extent Backup (page 259)
- How To Restore from an Online Extent Backup (page 259)

  An online extent backup is, essentially, a snapshot copy of the repository extents with the system running. Online extent backup is the primary means of performing repository backups and can be run during production hours.
- How To Make a Smalltalk Full Backup (page 264)
- How To Restore from a Smalltalk Full Backup (page 269)

  As an alternative, you can perform full backups using backup and restore methods provided as part of the GemStone kernel. Full backups are required if you want to reduce the number of extents in the repository.

- How To Make an Offline Extent Backup (page 287)
- How To Restore from an Offline Extent Backup (page 287)

  It is also possible to backup and restore using offline extent copy, using a snapshot of the physical extent files obtained while GemStone is shut down.
- How To Recover After Repair of the File System (page 259)

  This section presents procedures for recovery from a disk failure or corrupted file system.

  NOTE

  To ensure protection from disk failure, we recommend that you either use mirrored disks or operating system mirroring. For more information, see "Developing a Failover Strategy" on page 34.
- Warm Standby Systems (page 294)

  To maximize system availability, you can run a second GemStone installation on standby, ready to take over in case of any failure of the primary system.

9.1 Overview

One way of safeguarding your repository is to create a GemStone backup periodically and then store the backup in a secure place. It's best to establish a regular backup schedule that fits your application and to keep system users informed of that schedule.

Also back up transaction logs, especially if you have enabled full transaction logging. These logs allow you to roll forward from a backup to the state of the last committed transaction. Transaction logs in the file system can be backed up as part of a regularly scheduled system backup using operating system utilities. For a related discussion, see "To Archive Logs" on page 250.

Because backup files do not include the object table, they are typically 15–20% smaller than the running repository, not including any free space in the extents.

Verify Backup Process

Creating a backup and archiving transaction logs is only useful if you can restore them successfully in case of a system failure. To make sure that your systems for archiving and restoring backups is complete and correct, it is good practice to perform the restore operation into a non-production system, replay tranlogs, and audit the repository.

Performing this exercise ensures that if you do have an emergency situation, you will have the required files available and be familiar with the process of restore, and avoid the risk of losing data.

9.2 How To Make an Online Extent Backup

Privileges required: SystemControl.

An online extent backup is, essentially, a snapshot copy of the repository extents with the system running. Online extent backup is the primary means of performing repository backups and can be run during production hours.

Three steps are involved in an online extent backup:

1. Suspend checkpoints.

   Checkpoints are not permitted while the online backup is in progress. There must not be a checkpoint in progress when the online backup begins, and no checkpoints are allowed to begin until the online backup has finished. All other database operations (including commits and aborts) are permitted during the online backup.

   To suspend checkpoints for a specified number of minutes, call `System class >> suspendCheckpointsForMinutes:`. If this method is called while a checkpoint is in progress, it will block until the current checkpoint completes. If one session attempts to suspend checkpoints and is blocked while the current checkpoint completes, and then a second session attempts to suspend checkpoints, the second session fails and the method returns false.

   You cannot suspend checkpoints while in partial transaction log mode or while the repository is in restore mode. However, you can start a new transaction log while checkpoints are suspended. To do so, call `Repository >> startNewLog`.

   To query the current status of checkpoints, call `System class >> checkpointStatus`. This method returns an Array object containing a Boolean that indicates whether checkpoints are suspended as well as an Integer that indicates the number of seconds remaining in the suspension.

For example:

```
topaz 1> printit
System checkpointStatus
%
an Array
  #1 false
  #2 0

topaz 1> printit
System suspendCheckpointsForMinutes: 15
%
true

topaz 1> printit
System checkpointStatus
%
an Array
  #1 true
  #2 900
```

2. Copy the repository extents.

   Once checkpoints are suspended, the session requesting the suspension can log out from GemStone and start the extent copy, using operating system commands.

3. Resume checkpoints.

   Once the extent copy has completed, a session will log in to GemStone and request the Stone to resume checkpoints (`System class >> resumeCheckpoints`). The result of this method is false if checkpoints were not previously suspended before executing `System class >> suspendCheckpointsForMinutes:` (as in step 1), and true if they were previously suspended.

   ```
   topaz 1> printit
   System resumeCheckpoints
   %
   true
   ```

   From this result, you can determine if the online extent backup was completed while checkpoints were still suspended. If the backup was completed in time, no further action is required and the backup is complete. If the backup did not

complete before checkpoints were resumed, then the backup must be discarded and another online extent backup must be taken.

If the system is shut down while checkpoints are suspended, checkpoints will be re-enabled and a final checkpoint will be written during the clean shutdown process. Any online backups in progress during system shutdown must be discarded.

Methods To Perform Checkpoints

To support the use of checkpoints, the following methods in class System (category Online Backup Support) are provided:

`suspendCheckpointsForMinutes:` *minutes*

Suspends all checkpoints for the given number of minutes or until the `resumeCheckpoints` method is executed, whichever occurs first. Calling this method while checkpoints are already suspended changes the duration of the suspension. If a checkpoint is in progress when this method is called, the call blocks until the current checkpoint completes, at which time checkpoints are suspended. If any session has made this call and is waiting for the current checkpoint to complete, calls to this method by other sessions will fail. Returns true if checkpoints were successfully suspended.

`resumeCheckpoints`

Resumes regular checkpoints if they were previously suspended by `suspendCheckpointsForMinutes:`. Returns true if checkpoints were previously suspended, returns false if checkpoints were not suspended.

`checkpointStatus`

Returns an Array of two elements: a Boolean that indicates if checkpoints are currently suspended, and an Integer indicating the number of seconds before checkpoints will be resumed by the stone.

In addition, two methods in class System (category Transaction Control) are provided for starting checkpoints:

`startCheckpointAsync`

Starts a new checkpoint if a checkpoint is not already in progress and returns immediately, without waiting for the checkpoint to finish. Returns true if a checkpoint is in progress or was started; returns false if a checkpoint could not be started because transaction logs are full or checkpoints are suspended. Does not commit, abort, or otherwise modify the current transaction.

`startCheckpointSync`

Starts a new checkpoint and returns when the new checkpoint has completed. If a checkpoint is already in progress, this method waits until the current checkpoint completes, then starts a new checkpoint. Returns true if a new checkpoint was successfully completed, returns false if a new checkpoint could not be started because transaction logs are full or checkpoints are suspended. Does not commit, abort, or otherwise modify the current transaction.

An Example Script

As part of a complete backup strategy, you can create an online extent backup script for your system. Your GemStone installation directories include the example script `$GEMSTONE/examples/admin/onlinebackup.sh`. You can customize this script for your own system; you must add the necessary code to perform the file system copies of your extents.

NOTE

The example script `onlinebackup.sh` *is unsupported. It is provided here for your convenience, and is subject to change in future releases.*

Be sure to review and test your script adequately to ensure the integrity of your backups.

As shown in `onlinebackup.sh`, your script must perform the following actions:

1. Suspend checkpoints. For example:

```
topaz 1> printit
System suspendCheckpointsForMinutes: minutes
%
```

2. Perform a file system backup of live extents.

   The actual backup may be as simple as using the UNIX `cp` command for each of the extent files. However, on production systems, this may involve breaking a disk mirror and subsequently resynchronizing. The example script does not include code for this step.

3. Resume checkpoints after the backup.

```
topaz 1> printit
System resumeCheckpoints
%
```

9.3 How To Restore from an Online Extent Backup

To restore the repository from an online extent backup to the last committed transaction, the online backup files must be available, along with all transaction logs written since the backup was started. (The previous transaction log may also be required.)

The restore procedure includes these steps:

1. Copy the extents from the backup to the location where the repository extents reside. This is essentially the reverse of Step 2 on page 260.
2. Use **startstone -R -N** to restart GemStone. These options start the Stone in restore mode, but do not attempt to automatically recover by replaying all transaction logs. (For more about these options, see page 411.)
3. Restore all transaction logs written since the online extent backup was performed. You can query the Stone to determine the sequence number of the first transaction log required for the restore.

   Each restore operation, on completion, terminates its GemStone session. You will need to log in again before performing the next restore operation.

   ```
   topaz 1> printit
   SystemRepository restoreStatusOldestFileId
   %
   6

   topaz 1> printit
   SystemRepository restoreFromCurrentLogs
   %
   Restore from transaction log(s) succeeded.
   ```

4. When all transaction logs have been restored, commit the restore.

   ```
   topaz 1> printit
   SystemRepository commitRestore
   %
   Restore from transaction log(s) succeeded. commitRestore
   succeeded
   ```

5. The repository is now ready for use.

9.4 How To Make a Smalltalk Full Backup

Privileges required: FileControl.

As an alternative to online extent backups, you can perform Smalltalk full backups during non-production hours, using backup and restore methods provided as part of the GemStone kernel. Smalltalk full backups are required if you want to reduce the number of extents in the repository or redistribute objects within the repository. During a Smalltalk full backup, dynamic internal data structures are copied and will be restored, improving performance of such routine maintenance tasks as garbage collection.

In a Smalltalk full backup, the method `Repository>>fullBackupTo:` saves the most recently committed version of the repository in a way that is consistent from a transaction viewpoint. The method `fullBackupTo:` forces a checkpoint of the repository at the time the method is executed and then creates a backup from that checkpoint, copying all objects in the repository and arranging them in a compact form. (An alternate method, `Repository>>fullBackupTo:mBytes:`, lets you create a multiple-file backup by limiting the size of each part.)

As with online extent backups, you can make Smalltalk full backups while the repository is in use. Other sessions can continue to commit transactions, but those transactions are not included in the backup.

With full transaction logging enabled, all work committed since the backup is saved in a set of transaction logs. In the event of a subsequent repository failure, the backup file together with the set of logs created since the backup contain all information necessary to restore the repository to its current committed state. In the absence of full transaction logging, a media failure can cause the loss of all updates since the last backup.

A Smalltalk full backup includes these three steps:

1. The Gem performing the backup scans the object table, building a list of objects to back up. This step runs in a transaction and can therefore cause a temporary commit record backlog in systems with high transaction rates. However, this step usually lasts ten minutes or less.
2. The Gem performing the backup next writes all shadow objects to the backup file. This step also runs in a transaction; furthermore, backing up shadow objects requires more disk I/O than backing up live objects, so the rate of objects backed up per second is slower in this step than in the next.

   (For definitions of shadow and live objects, see “Basic Concepts” on page 304.)

3. In the final step, all live objects are written to the backup file. This step is performed outside a transaction; if the Stone signals the session to abort, it will do so. For most systems, this step takes the longest of the three.

The fullBackupTo: Methods

`Repository>>fullBackupTo:`*filename*
`Repository>>fullBackupTo:`*filename* `MBytes:`*mByteLimit*

In each of these methods, *filename* specifies the file where the backup is to be created. You must specify the name of a file, not just a directory name. You may include a relative or absolute path in addition to the file name.

If you use a relative path, the path is relative to the directory of the Gem process or linked session. For linked topaz sessions, this is the directory from which topaz was started. For RPC Gems, this is either specified by #dir: in the login parameters, or the home directory of the Gem's UNIX user.

You can create backups on a remote node by using a network resource string (NRS) to specify the node name as part of *filename*. For an example, see "To Create a Backup on a Remote Node" on page 267.

WARNING

If the specified destination runs out of space, the backup will terminate with a system I/O error at that point, and will be unusable. To avoid having to repeat the entire backup, make sure that you have sufficient space or set mByteLimit *appropriately.*

mByteLimit lets you create a multiple-file backup by limiting the size of each part. For further information and an example, see "To Create a Backup in Multiple Files" on page 267. If you don't want to limit the size of the backup file, specify a *mByteLimit* of 0 or use the simpler `Repository>>fullBackupTo:` message, omitting `MBytes:` entirely. A value of *mByteLimit* less than 0 or greater than 4096000 generates an error.

To perform a Smalltalk full backup, send your repository the message `fullBackupTo:`. For example:

```
topaz 1> printit
"Create a full backup of SystemRepository in the file
      '/users/backups/mar15.11'"
SystemRepository fullBackupTo:'/users/backups/mar15.11'
%
true
```

This writes the backup file named **`mar15.11`**. Messages are written to the stone log indicating when the backup started and when it completed.

During the backup, the session is put into manual transaction mode so the backup won't interfere with ongoing garbage collection. When the backup completes, the session is left outside of a transaction. If you want to make changes to the repository after a backup, send `System beginTransaction` or `System transactionMode: #autoBegin`.

If the backup file already exists, or if the *filename* argument is an empty string, the method returns an error.

Additional Performance Tips

For the session performing the backup, the statistic ProgressCount (described on page 509) indicates the number of objects written to the backup file thus far. If you know the number of objects in the repository, you can use this statistic to determine how far the backup has progressed.

You can often improve both backup and restore performance by increasing the size of the shared page cache.

Backups and Garbage Collection

NOTE

You will find it easier to understand the following discussion if you have first read and understood the section "Basic Concepts" on page 304.

Because shadow objects must be backed up, it is more efficient to run a Smalltalk full backup when there are few shadow objects. If possible, first check the statistic PagesNeedReclaimSize (page 505). If that statistic is high, run one or more Reclaim GcGems before performing the backup. (See "Admin and Reclaim GcGems" on page 311.)

Dead objects waiting to be reclaimed (measured by the statistic DeadNotReclaimedKobjs, described on page 483) are not backed up, as these objects are going to be deleted anyway.

Possibly dead objects are included in the backup file. (Possibly dead objects are defined in "What Happens to Garbage?" on page 309). However, the possible dead set is not backed up. So if a markForCollection or other garbage-marking operation completed before the backup, but the possibly dead objects had not yet been promoted to dead, the garbage-marking operation will have to be repeated.

To avoid this, therefore, if you back up your repository after a markForCollection or other garbage-marking operation, wait until the statistic PossibleDeadObjs (page 507) falls, and the statistic DeadNotReclaimedObjs (page 483) rises.

To Create a Backup on a Remote Node

The following example uses an NRS to create a backup on a disk on another node. Of course, performing a backup across the network is likely to take much longer than writing it to a local device.

```
topaz 1> printit
SystemRepository
      fullBackupTo:'!@flute!/gemstone/backups/mar15.11.dbf'
%
```

A GemStone NetLDI must be running on the remote node. The user performing the backup must provide authentication for that node, such as an entry in a `.netrc` file. The requirements are similar to those given in Chapter 3 for starting an RPC Gem session process on a remote node.

To Create a Backup in Multiple Files

To create a multiple-file backup, include the argument `MBytes:`*mByteLimit.* For example, `MBytes:100` tells GemStone to limit the size of the backup file to 100 MB. Writing a backup to multiple files takes longer than writing it to one file.

If a backup requires more space than you specified in *mByteLimit*, the backup stops after creating one file and returns a result similar to this:

```
topaz 1> printit
" Start a full backup of SystemRepository "
SystemRepository fullBackupTo: '/users/backups/mar15.11'
      MBytes: 50
%
Finished writing backup file 0 of a multifile backup
```

To create the next file in the backup, send your repository the message `continueFullBackupTo:`*fileOrDevice* `MBytes:`*mByteLimit.*

For example:

```
topaz 1> printit
" Continue a full backup, writing a second file "
SystemRepository continueFullBackupTo:
      '/users/backups/mar15.11_2' MBytes: 50
true
```

If the backup is suspended because it reached the specified byte limit, the session may or may not be in a transaction, depending on how far the backup has progressed. The `continueFullBackupTo:` method operates properly in either case.

Commits and aborts by the session doing a multiple-file backup are disallowed until the backup completes. If you need to cancel the backup, use the method `Repository >> abortFullBackup.` The session can then commit or abort its current transaction.

When backing up to multiple disk files, be sure to specify a unique file name for each continuation. If you need to verify the file sequence later, each file contains a sequential fileId, which you can examine using **copydbf -i** *fileName*.

To Create Compressed Backups

It is possible to write and read full backup files in compressed mode.

Writing to, and reading from, a compressed file can be performed only to a local file system file or to a file system that is NFS-mounted.

Backup files written in compressed mode are automatically appended with the suffix `.gz` if you do not specify that suffix and if the backup is being written to a file system file.

All restore methods automatically detect whether a file is compressed or not and read the file accordingly. Even a backup originally created in uncompressed mode, then later compressed externally with gzip, is readable by `restoreFromBackup:`.

NOTE

*Transaction logs are always written in uncompressed format. These files may be compressed with **gzip** before archiving them. Such archived tranlogs can be restored directly, without having to run **gunzip** on them, although the process is less efficient.*

To support compression, the following methods in class Repository (category Backup and Restore) are provided:

`continueFullBackupCompressedTo:` *fileOrDevice* `MBytes:` *mByteLimit*

This method is similar to `continueFullBackupTo:MBytes:` except that the output file is written compressed in gzip format. The output file must be on a local file system or accessible via NFS. Backup files written to a file system in compressed mode are automatically appended with the suffix `.gz` if that suffix is not specified by the user.

`fullBackupCompressedTo:` *fileOrDevice*

This method backs up the receiver to a single backup file in gzip format. The output file is written compressed in gzip format and cannot be written to a raw device. The output file must be on a local file system or accessible via NFS. Backup files written to a file system in compressed mode are automatically appended with the suffix `.gz` if that suffix is not specified by the user.

`fullBackupCompressedTo:` *fileOrDevice* `MBytes:` *mByteLimit*

This method is similar to `fullBackupTo:MBytes:` except that the output file is written compressed in gzip format. See `fullBackupCompressedTo:`

To Verify a Backup is Readable

To verify that a backup file is readable, use the GemStone utility **copydbf**. You can conserve disk space and reduce disk activity by specifying `/dev/null` as the destination. For instance:

```
% copydbf /users/backup/mar15.11 /dev/null
```

Checking Backup Start and Completion

The time a backup is started, and the time that it completes successfully, are written to the stone log.

9.5 How To Restore from a Smalltalk Full Backup

Privileges required: FileControl.

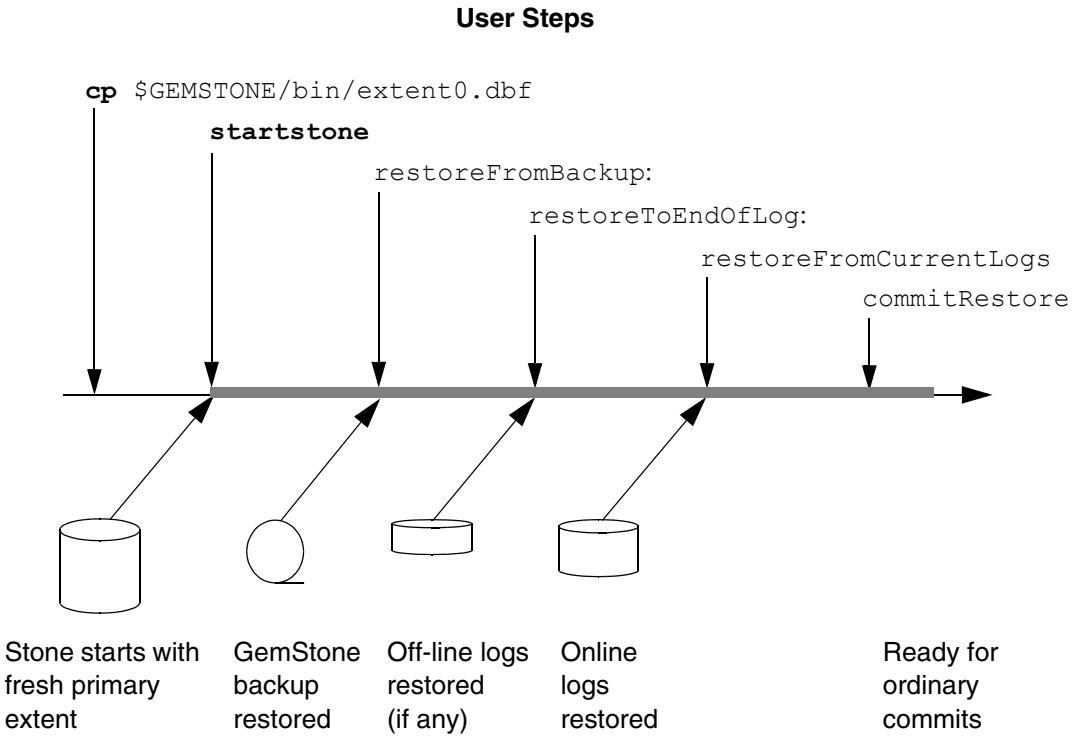
Restoring the repository from a Smalltalk full backup ordinarily takes place in two phases:

1. Restore the repository from the last GemStone backup file.
   - For a step-by-step description, see “Phase 1: Restore to the Point of the Backup” on page 271.

2. Apply transaction logs to restore transactions that were committed after the backup was started. (The backup must have been made while the repository was in full transaction logging mode.)
    - For a step-by-step description, see "Phase 2: Restore Subsequent Transactions" on page 274.

Figure 9.1 illustrates the restore process for Smalltalk full backups.

Figure 9.1 System Timeline: Restoring from a Smalltalk Full Backup



Before you begin, make sure you have a Smalltalk full backup of the system repository, created by GemStone with the method `fullBackupTo:`. (See "How To Make a Smalltalk Full Backup" on page 264.)

If full backups of your repository require more than one backup file, restoring the first and not all the later ones does not give you a usable version of the repository. The objects in the first backup file are copied, but the transaction cannot be committed until you restore the remainder of the backup.

After the backup has been restored, the repository reflects its state at the time of the backup. All the objects are intact and ordinarily are clustered in a way similar, but not identical, to their organization in the original repository. This clustering reflects both explicit clustering of objects by the application and default clustering into the generic "don't care" cluster bucket.

If the number of extents during restoration is the same as when the backup was started, such clustering in the original repository takes precedence over the DBF_ALLOCATION_MODE configuration setting used by the Stone performing the restore operation. If the number of extents differs, then the DBF_ALLOCATION_MODE setting at the time of the restore controls the distribution of objects across extents.

Phase 1: Restore to the Point of the Backup

To begin, you need a file copy (*not* a GemStone backup) of a good repository. We recommend that you use a copy of the `extent0.dbf` that was shipped in `$GEMSTONE/bin`, although any extent file that is a complete, uncorrupted repository will work. If you are using the backup/restore process to reduce the size of your extent, the new extent file must be smaller than your current extent.

NOTE

Make sure that you have full backups of good repository files. If the backup consists of multiple files, the complete set must be available.

The user restoring the backup must be the only user logged in to the server. The method that starts the restoration will suspend other logins.

NOTE

We recommend that you log in as DataCurator or SystemUser to restore the backup. If you start the restore as another user and that UserProfile disappears as a result of the restore, Topaz will see a fatal error.

To restore your repository from a Smalltalk full backup, perform the following procedure:

Step A1. If GemStone is still running, tell all users to log out and use **stopstone** to stop the system. Certain file system failures while the Stone is running may make it necessary to use **kill** *processid* to kill the Stone process.

Step A2. If you are restoring the repository because of a suspected GemStone failure, preserve a copy of the extents in case Technical Support wants to examine them.

WARNING
Do NOT delete the transaction log files up to the time of the crash – leave them online in their current locations.

Step A3. Delete all extent files specified in DBF_EXTENT_NAMES in your configuration file.

Step A4. Copy the distribution extent to the location of your primary extent, which is the extent listed first in DBF_EXTENT_NAMES.

Make sure there are no other extent files in that location. Do not copy any other extent files to the location of the *new (second)* extent; the Stone repository monitor will create the new extent at startup.

We recommend that you use the GemStone **copydbf** command to create the copy, rather than using the UNIX **cp** command; **copydbf** must be used if you are copying to or from a raw partition.

For example:

```
% copydbf $GEMSTONE/bin/extent0.dbf \
  $GEMSTONE/data/extent0.dbf
% chmod 600 $GEMSTONE/data/extent0.dbf
```

Use **chmod** to give the copy the same permissions you ordinarily assign to your repository files.

Step A5. Ensure that there is space to create a log file during recovery. At least one of the directories specified by STN_TRAN_LOG_DIRECTORIES must have space available or one of the raw partitions must be empty. You may need to add entries to STN_TRAN_LOG_DIRECTORIES and STN_TRAN_LOG_SIZES in your configuration file.

GemStone starts a new transaction log file when you start the restore process (Step A8). In addition, you can force closure of the current log and opening of

a new log at any time by using the method `Repository>>startNewLog`. That method returns a SmallInteger, which is the *fileId* of the new log.

Step A6. Use **startstone** to restart the Stone.

For optimal performance, your extent files should be pre-grown during startup, rather than growing incrementally during restore. See "Pregrowing Extents to a Fixed Size" on page 45.

Step A7. Log in to GemStone as DataCurator or SystemUser using linked Topaz (**topaz -l**). Remember that the password will be the original one supplied when you installed GemStone, not necessarily the one you have been using.

NOTE

To perform the following steps, you must be the only user logged in to GemStone. Once you start the next step, other logins will be suspended.

Step A8. Restore the most recent full backup to the new repository by sending the message `restoreFromBackup:` *fileOrDevice*. This method automatically detects whether a backup is compressed or not and reads it accordingly.

The following example restores the repository from a backup that consists of a single disk file. For information about restoring backups from more than one file, see "To Restore Multiple-File Backups" on page 278.

The message `restoreStatus` can return helpful information at any point in the restore process. This status is an attribute of the repository, not of the session, and persists across login sessions and stopping and starting of the Stone repository monitor.

```
topaz 1> printit
SystemRepository restoreStatus
%
Restore is not active
topaz 1> printit
SystemRepository restoreFromBackup: 'backup.dat'
%
```

If the full backup is contained in one file, the system commits the restore and returns a summary and status. For instance:

```
[restore info]:Reading from backup.dat.

[Info]: Logging out at 05/17/11 20:37:56 PDT
The restore from full backup completed, with 97470
objects restored.
 Ready for restore from transaction log(s).
```

- If partial logging was in effect (STN_TRAN_FULL_LOGGING = false) at the time the backup was made, the final status line reads:

  ```
  Restore complete. (Backup made while in partial
  logging mode.)
  ```

  This status means that transaction logs cannot be restored. The repository is ready for ordinary use, and logins have been enabled.

- If full logging was in effect (STN_TRAN_FULL_LOGGING = true), the status line indicates the next step:

  ```
  Ready for restore from transaction log(s).
  ```

  Continue with "Phase 2: Restore Subsequent Transactions" on page 274.

  CAUTION

  Although you can end the restore process before restoring from all transaction logs, doing so can make it impossible to restore the omitted logs later by repeating the process. If you plan to terminate the restore prematurely, first read "Precautions When Restoring a Subset of Transaction Logs" on page 283.

Phase 2: Restore Subsequent Transactions

If full transaction logging was in effect, the second phase of restoring the repository is to roll forward from the state of the last backup to the state of the last committed transaction. This action repeats the transactions in the order in which they were committed. You can do this only if the STN_TRAN_FULL_LOGGING configuration option was set to True at the time the backup was made. You can only restore transactions committed within a single backup-restore cycle; that is, the transactions being restored cannot span a more recent restore.

CAUTION

Ordinarily, you will restore transactions from all log files written since

the backup. If for some reason you plan to omit one or more log files, first read "Precautions When Restoring a Subset of Transaction Logs" on page 283.

Step B1. Before continuing the restore process, you must log in again. (The `restoreFromBackup:` method (Step A8) terminated the session when it completed.)

```
topaz> login
```

Step B2. Determine the location of all needed transaction logs. The method `restoreStatus` identifies the oldest transaction log that is needed, or you can use **copydbf -i** *backupName*. If a session was in a lengthy transaction at the time of the backup, that log may be one that was written before the backup started. In this example, it is `tranlog6.dbf`:

```
topaz 1> printit
SystemRepository restoreStatus
%
Restoring from Transaction Log files,
  restored to 03/02/11 13:51:19 PST
  next fileId = 6, record = 9.
```

Compare the fileId in the message with the names of the transaction log files in the directories specified in STN_TRAN_LOG_DIRECTORIES. For transaction logs in the file system, fileId forms the numeric portion of the file name, tranlog*NN*.dbf. For transaction logs in raw partitions, use **copydbf -i** *fileName* to display the fileId.

The methods in category Configuration File Access of class System allow you to inspect the log directory paths and file name prefixes for the current configuration. (For information, see "How To Access the Server Configuration at Run Time" on page 62.)

Step B3. Restore the transaction logs in time sequence, beginning with the log identified by `restoreStatus`. How you restore the logs depends on where the oldest logs are located. If you encounter a failure because of a truncated or corrupted transaction log, refer to "Errors While Restoring Transaction Logs" on page 280.

- ❒ If all transaction log files beginning with `tranlog`*fileId*`.dbf` are in the locations specified by the STN_TRAN_LOG_DIRECTORIES configuration option, skip to Step B4.

- ❒ If several of the older transaction logs have been moved to a different disk location, send the messages `Repository>>setArchiveLogDirectories:` and `Repository>>restoreFromArchiveLogs` to restore those log files. The following example restores logs archived in `/GS-archive`, which is not one of the active locations listed in STN_TRAN_LOG_DIRECTORIES:

  ```
  topaz 1> printit
  SystemRepository setArchiveLogDirectories:
    #( 'GS-archive' )
  SystemRepository restoreFromArchiveLogs
  %
  ```

  See the method comments in the image for details. A directory location can include an NRS for a remote node, but a NetLDI must be running on that node.

  Continue with the next substep if it applies; otherwise continue with Step B4.

- ❒ If individual transaction logs need to be restored, execute `Repository>>restoreToEndOfLog:` *fileId* for each file or raw partition in time sequence. For example:

  ```
  topaz 1> printit
  SystemRepository restoreStatus
  %
  Restoring from Transaction Log files,
    restored to 03/02/11 13:51:19 PST
    next fileId = 7, record = 10. oldest fileId = 7
  topaz 1> printit
  SystemRepository restoreToEndOfLog: 7
  %
  [Info]: Logging out at 03/24/11 14:37:07 PDT
  Restore from transaction log(s) succeeded.
  ```

  As mentioned previously, each restore operation, on completion, terminates its GemStone session. You will need to log in again before performing the next restore operation.

  Continue with Step B4 to restore from online logs.

Step B4. Restore transactions from all remaining online log files by executing the method `Repository>>restoreFromCurrentLogs`. The remaining log files (all log files beginning with the *fileId* currently returned by

restoreStatus) must be online and in the directories or raw partitions specified by STN_TRAN_LOG_DIRECTORIES.

Login, then:

```
topaz 1> printit
SystemRepository restoreFromCurrentLogs
%
[Info]: Logging out at 03/24/11 14:37:07 PDT
Restore from transaction log(s) succeeded.
```

Step B5. Send the message commitRestore, which tells the system you are finished restoring transaction logs.

Login, then:

```
topaz 1> printit
SystemRepository commitRestore
%
[Info]: Logging out at 03/24/11 14:37:07 PDT
Restore from transaction log(s) succeeded, commitRestore
succeeded.
```

At this point, restore mode is no longer active, and no further logs can be restored. Logins have been enabled, and ordinary user commits will be allowed.

If you send commitRestore earlier in the restore process (prior to restoreFromCurrentLogs), a warning is issued because all previously committed transactions may not have been restored. However, this usage provides a way to recover as much as is available when a log file has been corrupted or lost.

This step completes the restore process. Make a new full backup as soon as operational circumstances permit.

Other Considerations

Performance Tips

Restoring from backup usually takes at least 10-30% longer than the full backup took.

For the session performing the restore, the statistic ProgressCount (described on page 509) indicates the number of objects restored from the backup file thus far. If

you know the number of objects in the backup file, you can use this statistic to determine how far the restore has progressed.

You can improve restore performance by using one page server for every extent that resides on its own dedicated disk drive, but only if the repository uses weighted extent allocation mode. For example, if you are restoring a repository with six extents, each on its own separate physical hard drive, the following parameters, set in the restoring Gem's configuration file for the duration of the restore, would improve performance:

```
STN_NUM_LOCAL_AIO_SERVERS = 6;
DBF_ALLOCATION_MODE = 10,10,10,10,10,10;
```

Here are some tips on getting restores completed as soon as possible:

- Use `restoreFromBackups:` or `restoreFromBackupsNoShadows:` methods, rather than restoring backups one by one. These methods require you to specify all backup files in the correct order in an Array.
- A large shared page cache will improve performance.
- Set the Gem's free frame limit to be very low (for example, 500).
- Restore from uncompressed files rather than compressed files. The decompression libraries are slow and do very inefficient I/O. Backups created from `fullBackupCompressedTo:` can be uncompressed using **gunzip** *before* the restore operation for best performance. This makes a big difference.
- Place extents on striped file systems.
- Make sure the backup files being restored are not on the same disk spindles as the extents or transaction logs.

To Restore Multiple-File Backups

If you are restoring a backup that occupies more than one file, each backup file must be restored in sequence. You can determine the restore status and the internal identification of the backup file required next by using the method `restoreStatus`. If a restore process is active, the reply indicates the date and time to which the repository has been restored, and the type of file and the internal file identification number (*fileId*) of the file that must be restored next. (The *fileId* is reset to 0 at the beginning of each Smalltalk full backup.)

For each part of the backup, repeat Step A8 on page 273, using the next file in sequence. If you are uncertain of the sequence in which to restore a particular file, use **copydbf -i** *fileName* to display its fileId.

Another way is to use `restoreFromBackups:` *arrayOfFilesOrDevices* and include all files or devices in the array in proper sequence. Since all backup files are verified for integrity at the beginning of the restore, they all must be available when this method is called. The following example restores a backup that occupies two files by placing the filenames in an array:

```
topaz 1> printit
SystemRepository restoreFromBackups:
#( '%/backups/mar15.11.1'
   '%/backups/mar15.11.2' )
%
Opening file %/backups/mar15.11.1
Opening file %/backups/mar15.11.2
Restore from full backup completed with 132804 objects
restored and 0 corrupt objects not restored.
```

Another alternative is to use `restoreFromBackupsNoShadows:` *arrayOfFilesOrDevices*. This method is the same as `restoreFromBackups:` except that partially filled data pages are not added to the list of scavengeable pages upon completion of the restore. Using this method may cause the restored repository to perform faster than one restored using `restoreFromBackups:`, especially immediately following the restore.

When you restore the last backup file, GemStone either commits the restore or indicates that it is ready to restore from transaction logs, as explained on page 274.

If an error occurs, the shadow object space being built by the restore reverts to its state as of the end of the last file that was successfully restored.

If you need to cancel the process and start over while restoring a multiple file backup, use the method `Repository>>abortRestore`.

If the Topaz login session dies before the last backup file has been restored, you must start over by restoring the first file of the set.

To Restore Logs to a Point in Time

Ordinarily, the methods `Repository>>restoreToEndOfLog:` and `restoreFromCurrentLogs` restore all transactions in the log file. However, you can specify an earlier stopping point by sending the message `timeToRestoreTo:` *aDateTime*. Restoration will stop at the first repository checkpoint that originally occurred at or after *aDateTime*.

To display the time a transaction log was started and the time of each checkpoint recorded in it, use **copydbf -I** *fileName*. By default, the maximum interval between checkpoints is five minutes. For example:

```
% copydbf -I tranlog2.dbf

Source file: tranlog2.dbf
  file type: tranlog  fileId: 2
  byteOrder: Sparc (MSB first) compatibilityLevel: 900
  The file was created at:      03/25/11 14:55:59 PDT.
  The previous file last recordId is  69.
  Scanning file to find last checkpoint...
Destination file:  /dev/null
     byteOrder: Sparc (MSB first)
   Checkpoint 1 started at: 03/25/11 14:55:59 PDT.
     oldest transaction references fileId -1 ( this file ).
   Checkpoint 2 started at: 03/25/11 14:57:23 PDT.
     oldest transaction references fileId -1 ( this file ).
   File size is 2.2 MBytes (4350 records).
```

The following sequence restores the repository to the first checkpoint that would have included a commit on March 22, 2011 at 2:56:00 p.m.:

```
topaz 1> printit
SystemRepository timeToRestoreTo:
 (DateTime fromString: '22/03/2011 14:56:00') .
SystemRepository restoreFromCurrentLogs
%
```

You can continue restoring past *aDateTime* by issuing another `restoreToEndOfLog:` or `restoreFromCurrentLogs`. If you first issue another `timeToRestoreTo:`, restoration stops at the new *aDateTime*; otherwise, restoration continues to the end of the log.

Errors While Restoring Transaction Logs

Sometimes a transaction log is inadvertently truncated or corrupted. This section discusses types of errors that can occur.

Disk-Full Error During a Copy Operation

An unnoticed disk-full error during a copy operation can result in a truncated log that goes undetected until you try to restore from it. Because of the way transaction logs are written, a truncated log may appear to restore properly, and the gap will not be detected until the next log is read.

If a message like the following appears:

```
Log with fileId 7 is truncated or corrupt, or log 8 is
corrupt.
```

check the *fileId* in the message to identify the log that caused the problem, which may be either the log currently being restored or the previous one. If the transaction log is a file (not a raw partition), its default name is `tranlog`*fileId*`.dbf`.

The method `restoreStatus` returns additional information indicating the next record expected within the current log file. For instance:

```
topaz 1> printit
SystemRepository restoreStatus
%
Restoring from Transaction Log files,
  restored to 03/02/11 13:26:31 PST
  next fileId = 6, oldest fileId = 6
```

Retrieve a copy of the transaction log from an OS backup, and complete restoring it by using `restoreToEndOfLog:`*fileId*. Then continue the restore process by using either `restoreFromCurrentLogs` or another `restoreToEndOfLog:`.

Missing Transaction Log File

If a transaction log file in the sequence is missing, `restoreFromCurrentLogs` stops at that point. For example, if `tranlog4.dbf` is missing, `restoreFromCurrentLogs` stops after restoring from `tranlog3.dbf`.

You can execute the method `restoreStatus` to identify the next log file expected. Ensure that the missing file(s) are present and then continue the restore process with `restoreFromCurrentLogs` or `restoreToEndOfLog:`.

In the following example, `restoreFromCurrentLogs` encounters a truncated log. Invoking `restoreStatus` confirms that `tranlog6.dbf` is incomplete and shows that restoration needs to continue with record 9:

```
topaz 1> printit
SystemRepository restoreFromBackup: 'backup.dat'
%
[restore info]:Reading from backup.dat.
The restore from full backup completed, with 40483 objects
restored.
Ready for restore from transaction log(s).
```

<Login>

```
topaz 1> printit
SystemRepository restoreFromCurrentLogs
%
[Info]: Logging out at 03/24/11 14:37:07 PDT
ERROR 4049 , Restore from transaction log failed
    Log with fileId 6 is truncated or corrupt, or log 7 is
corrupt.
```

<Login>

```
topaz 1> printit
SystemRepository restoreStatus
%
Restoring from Transaction Log files,
  restored to 03/02/11 13:26:31 PST
  next fileId = 6 , oldest fileId = 9
```

To recover, restore a complete copy of that transaction log by name (here, "tranlog6.dbf"):

```
topaz 1> printit
SystemRepository restoreToEndOfLog: 6
%
[Info]: Logging out at 03/24/11 14:37:07 PDT
Restore from transaction log(s) succeeded.
```

<Login>

```
topaz 1> printit
SystemRepository restoreStatus
%
Restoring from Transaction Log files,
  restored to 03/02/11 13:51:19 PST
  next fileId = 7.
```

The restore status now shows that you are ready for the next transaction log, `tranlog7.dbf`. Because the remaining logs are online, return to the original procedure of restoring them as a group and then commit the restored repository:

```
topaz 1> printit
SystemRepository restoreFromCurrentLogs
%
[Info]: Logging out at 03/24/11 14:37:07 PDT
Restore from transaction log(s) succeeded.

<Login>

topaz 1> printit
SystemRepository commitRestore
%
Restore from transaction log(s) succeeded., commitRestore
succeeded
```

If you cannot find an undamaged copy of the transaction log, `commitRestore` will commit as much as has been restored. However, if there is any chance of a finding a good copy, see the following discussion,"Precautions When Restoring a Subset of Transaction Logs".

Precautions When Restoring a Subset of Transaction Logs

If, for some reason, you want to end the restore process before restoring transactions from one or more log files, be aware of the likely consequences and some precautions that may be appropriate:

- Obviously, the omitted transactions will be lost. Unless the omitted log is missing, presumably that is your intent.
- Less obvious, where the omitted logs actually are present, is that it may be difficult or impossible to reverse your action later and restore the omitted logs by repeating the entire restore process. Operations after the first restore create a time fork in the repository, and attempting to reverse the course later results in object audit errors. (For an example of this, see the following discussion, "Fork-in-Time" Scenario.)

If there is any chance that you may want to restore from the omitted transaction logs later, modify the ordinary restore procedure in this way:

1. Before starting the Stone on the fresh extent (Step A8, on page 273), move all transaction logs to another directory.
2. After restoring from the backup (`restoreFromBackup:`), restore transaction logs individually by using `restoreToEndOfLog:` (Step B2, on page 275) and providing the new path.

   Repeat `restoreToEndOfLog:` for each log file you want to restore.
3. Send the message `commitRestore` to end the restore operation.

CAUTION

If you subsequently decide to restore logs by repeating the entire process, first remove any new log files since the start of the restore process, and then move the previous transaction logs back to their original location. Follow the ordinary restore procedure (page 269). Note, however, that you will not be able to restore any transactions committed after your initial `commitRestore`*. That work will be lost.*

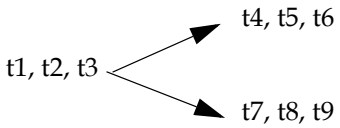
"Fork-in-Time" Scenario

In some cases, you may encounter problems when there you cannot restore from your most recent backup file and must restore from an earlier backup. This scenario presents a risk of transaction logs that are out of sequence due to a "fork-in-time." Consider the following sequence of repository events:

1. Generate backup1.
2. Generate transaction logs t1, t2, t3.
3. Generate backup2.
4. Generate transaction logs t4, t5, t6.
5. Restore backup2.
6. commitRestore (without replaying transaction logs t4, t5, t6).

   The repository is now at same state as step 3.
7. Generate transaction logs t7, t8, t9.
8. Restore backup1.
9. Replay transaction logs t1 through t9.

In terms of the repository lifecycle, this scenario has two timelines, with a fork-in-time at the end of t3:



If, at step 5, we also restored the transaction logs (t4, t5, t6), the resulting sequence could be replayed without problems. The problem is caused when the continuity of the transaction log chain is broken.

After restoring backup1 in step 8, it would be possible to safely replay transaction logs t1 through t6 without problems, but any changes made in (t7, t8, t9) would be lost.

During step 9, the replay of (t7, t8, t9) is likely to produce problems. If any object changes made in (t4, t5, t6) are logically inconsistent with those made in (t7, t8, t9), possible errors are wide-ranging, including UTL_ASSERT/UTL_GUARANTEE errors or errors of the form:

```
recovery/restore: invalid operation XXXXXXXXXX
Transaction expected to abort.
non-empty invalidObjs in recover.c:commitTran
```

In the worst case, errors may not be written to the Stone log during transaction log replay, but the final repository may be corrupted in obscure ways. If the corruption is structural, it may be detected by an object audit (page 165). Otherwise, the corruption may go undetected unless picked up by application code.

If you are presented with a situation wherein you are forced to restore from an earlier backup, keep in mind the following:

1. Be aware of the fork-in-time phenomenon and avoid restore/replay operations that would create a fork.
2. When restoring into an ongoing transaction log sequence, only restore a backup file generated earlier within that same sequence, and then replay *all* transaction logs in that sequence generated since that backup.
3. If for some reason you cannot follow guideline 2, realize that you cannot restore from an earlier backup and replay transaction logs beyond the point of the initially restored backup.

Methods for the Restore Process

To support the restore process, the following methods in class Repository (category Backup and Restore) are provided:

`restoreStatusOldestFileId`

This method returns the fileId of the oldest transaction log that needs to be present for the next restore from log operation, or nil if restore is not active.

`restoreToEndOfLog:` *fileId*

This method is used after a restore from an online extent backup or Smalltalk full backup, to restore all transaction logs up to and including the log with the given *fileId*. By default, the method reads log files from the directories specified by the Stone's STN_TRAN_LOG_DIRECTORIES configuration file parameter. If a `setArchiveLogDirectories:` or `restoreFromArchiveLogs` method has been executed since the previous **startstone** command, the method reads log files from those directories instead.

`setArchiveLogDirectories:` *arrayOfDirectorySpec*

This method specifies an Array of file system directories and/or raw devices to be used by subsequent invocation(s) of `restoreFromArchiveLogs` or `restoreToEndOfLog:`.

`setArchiveLogDirectory:` *aDirectorySpec*

This method specifies a single file system directory or raw device to be used by subsequent invocation(s) of `restoreFromArchiveLogs` or `restoreToEndOfLog:`.

`setArchiveLogDirectory:` *aDirectorySpec* `tranlogPrefix:` *aPrefix*

This method is similar to `setArchiveLogDirectory:` (above), except that the transaction log filenames begin with *aPrefix* rather than `tranlog`.

9.6 How To Make an Offline Extent Backup

To make a file system backup of the extents while GemStone is running, see page 259.

You cannot make a file system backup of a running GemStone system unless checkpoints are disabled. If you make an extent file backup while GemStone is running and checkpoints are enabled, that backup is likely to be unusable.

You can safely perform a file system backup of the extents while GemStone is shut down, provided that GemStone was shut down cleanly, such as by **stopstone** *gemStoneName*. During the shutdown process, a checkpoint is performed in which all committed transactions are written to the extents. A copy of the extents after an orderly shutdown constitutes a complete operating system backup of the repository without the necessity of backing up existing transaction logs.

To restore the repository using offline extent file backups, see the next section, "How To Restore from an Offline Extent Backup."

9.7 How To Restore from an Offline Extent Backup

Privileges required: FileControl.

This section describes how to restore the repository from an operating system backup made using **dump**, **tar**, **cp**, or similar methods. For complete recovery, this backup must have been made while the repository monitor was shut down. That is, the backup must have been started after a **stopstone** and must have been completed before the subsequent **startstone**.

If the file system itself has been corrupted, not just the extent files, see the section "Disk Failure or File System Corruption" on page 147.

NOTE

Before you begin, make sure that you have operating system backups of good extents and that the backups were made after an orderly shutdown of the Stone repository monitor. If a backup consists of multiple files, the complete set must be available.

Step 1. If GemStone is still running, tell all users to log out and use **stopstone** to stop the repository monitor. Certain file system failures while the Stone is running may make it necessary to use **kill** *processid* to kill the Stone process.

Step 2. If you are restoring the repository because of a suspected GemStone failure, preserve a copy of the extents in case Technical Support wants to examine them.

WARNING

Do NOT delete the transaction log files up to the time of the crash – leave them online in their current locations.

Step 3. Delete all extent files specified by DBF_EXTENT_NAMES in your configuration file.

Step 4. Restore the operating system backup copies of the extent files to the locations specified by the DBF_EXTENT_NAMES configuration option.

Step 5. Ensure that there is space to create a log file during recovery. At least one of the directories specified by STN_TRAN_LOG_DIRECTORIES must have space available or one of the raw partitions must be empty. You may need to add entries to STN_TRAN_LOG_DIRECTORIES and STN_TRAN_LOG_SIZES in your configuration file.

Step 6. The next step depends on whether full transaction logging was in effect at the time the backup was made.

- If partial transaction logging (STN_TRAN_FULL_LOGGING = False) was in effect at the time the backup was made, the restore process is complete. Restart GemStone by invoking **startstone** in the usual manner.
- If full transaction logging (STN_TRAN_FULL_LOGGING = True) was in effect, continue by restoring from transaction logs. Use **startstone -R** to restart GemStone.

  Continue with Step 7.

Step 7. Determine the location of all needed transaction logs. The method `restoreStatus` identifies the earliest transaction log that is needed. In this example it is `tranlog6.dbf`:

```
topaz 1> printit
SystemRepository restoreStatus
%
Restoring from Transaction Log files,
  restored to 03/02/11 13:26:31 PST
  next fileId = 6, record = 9.
```

Compare the fileId in the message with the names of the transaction log files in the directories specified in STN_TRAN_LOG_DIRECTORIES. For transaction logs in the file system, fileId forms the numeric portion of the file name, tranlog*NN*.dbf. For transaction logs in raw partitions, use **copydbf -i** *fileName* to display the fileId.

The methods in category Configuration File Access of class System allow you to inspect the log directory paths and file name prefixes for the current configuration (for information, see "How To Access the Server Configuration at Run Time" on page 62).

Step 8. Restore the transaction logs in time sequence, beginning with the log identified by `restoreStatus`. How you do restore the logs depends on

where the oldest logs are located and is described next. If you encounter a failure because of a truncated or corrupted transaction log, refer to "Errors While Restoring Transaction Logs" on page 280.

- If all transaction log files beginning with `tranlog`*fileId*`.dbf` are in the locations specified by the STN_TRAN_LOG_DIRECTORIES configuration option, skip to Step 9.
- If several of the older transaction logs have been moved to a different disk location, send `Repository>>setArchiveLogDirectories:` and `Repository>>restoreFromArchiveLogs` to restore those log files. The following example restores logs archived in `/GS-archive`, which is not one of the active locations listed in STN_TRAN_LOG_DIRECTORIES:

```
topaz> login
gci login: currSession 1 rpc gem processId -1
successful login
topaz 1> printit
SystemRepository setArchiveLogDirectories:
  #( 'GS-archive' )
SystemRepository restoreFromArchiveLogs
%
```

  See the method comments in the image for details. A directory location can include an NRS for a remote node, but a NetLDI must be running on that node.

  Continue with the next substep if it applies, or else with Step 9.

- If individual transaction logs need to be restored, execute `Repository>>restoreToEndOfLog:` *fileId* for each file or raw partition in time sequence. For example:

```
topaz> login
gci login: currSession 1 rpc gem processId -1
successful login
topaz 1> printit
SystemRepository restoreStatus
%
Restoring from Transaction Log files,
  restored to 03/02/11 13:51:19 PST
  next fileId = 7, record = 10.
topaz 1> printit
SystemRepository restoreToEndOfLog: 7
%
Restore from transaction log succeeded.
```

  Continue with Step 9 to restore from online logs.

Step 9. After you restore transactions from any offline log files, restore transactions from the online log files by executing the method `Repository>>restoreFromCurrentLogs`. All the remaining log files must be in directories or raw partitions specified in STN_TRAN_LOG_DIRECTORIES.

```
topaz> login
topazm: in login, found topaz session 1 stale
gci login: currSession 1 rpc gem processId -1
successful login
topaz 1> printit
SystemRepository restoreFromCurrentLogs
%
Restore from transaction log(s) succeeded.
```

Step 10. If restoration from the transaction logs was successful, send the message `commitRestore` to tell the system that you are finished restoring. No further logs can be restored, and normal user commits will be allowed.

```
topaz> login
topazm: in login, found topaz session 1 stale
gci login: currSession 1 rpc gem processId -1
successful login
topaz 1> printit
SystemRepository commitRestore
%
Restore from transaction log(s) succeeded. commitRestore
succeeded
```

Step 11. Make a new GemStone backup as soon as operational circumstances permit.

9.8 How To Recover After Repair of the File System

In case of a disk failure or a corrupt file system, the file system must be repaired before you can restart GemStone. The procedure you need to follow depends on how the damage was repaired.

To Recover After a File System Repair With fsck

After a repair with **fsck** (the UNIX file system consistency check and interactive repair utility), use **pageaudit** to check the condition of the system repository. (For instructions, see "How To Audit the Repository" on page 165.)

- If the page audit succeeds, try to restart GemStone again. If GemStone starts, you can resume normal operations.
- If the page audit fails or GemStone doesn't start, you will need to restore the repository file. (See "How To Restore from an Online Extent Backup" on page 263 or "How To Restore from a Smalltalk Full Backup" on page 269.)

To Recover When a File System Must Be Restored

If your system administrator intends to restore the file system from a backup device, it might be worthwhile to copy the repository to another node **before** restoring the file system. Although this copy may prove unusable, if a great deal of important data has been committed since the last backup, it may be worth a try.

To restart GemStone after the file system is restored, perform the following steps. (See Figure 9.2 on page 293.)

Step 1. If you made a copy of the repository, begin with that copy. To test the copy, use the methods discussed in the section "How To Audit the Repository" on page 165. You will need to specify the name and path of the copy using a temporary configuration file so that **pageaudit** is not performed on the extent that was restored along with the rest of the file system.

If you didn't make a copy of the repository or the copy does not pass **pageaudit**, start with the current extent file(s) that were restored from the file system backup. Review the log file and determine whether the backup was made while GemStone was running.

- ❒ If the Stone was running and checkpoints were not disabled and any changes were being made to the repository during the backup, the extent file(s) may be inconsistent and cannot be made to work. In that case, you must restore from a valid GemStone backup. However, the transaction logs should be usable.
- ❒ If the backup was done while GemStone checkpoints were suspended, continue to the next step.

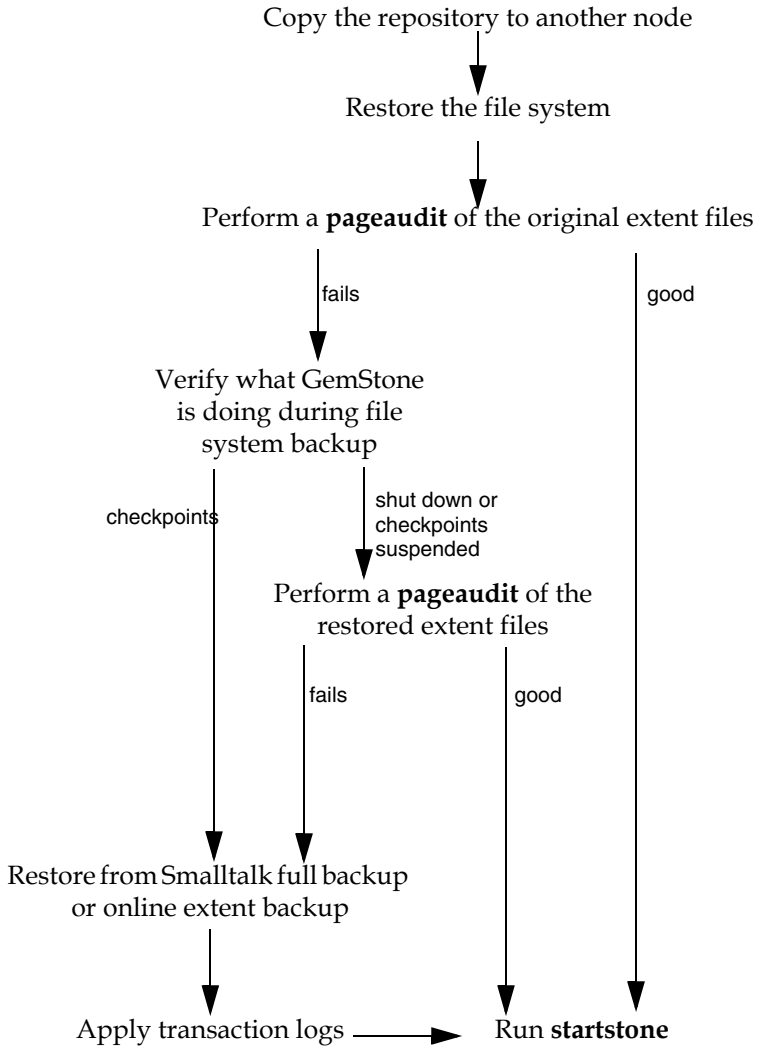
Step 2. Do a **pageaudit** to check the current (restored) extent files. (For instructions, see "To Perform a Page Audit" on page 165.)

- ❒ If the page audit is good, try to restart the system again with **startstone**. If GemStone starts, you can resume normal operations.
- ❒ If the page audit fails or GemStone doesn't start, you will need to restore from a GemStone online extent backup (page 263) or from Smalltalk full backups (page 269).

NOTE

Remember that **startstone** *uses (1) a GEMSTONE_SYS_CONF environment variable or (2) $GEMSTONE/data/system.conf as the default system-wide configuration file. If you want it to use parameters from a different configuration file, be sure to specify that file with the* **startstone -z** *option. For a discussion of how* **startstone** *searches for the configuration file, see page 350.*

Figure 9.2 Recovering When a File System Must Be Restored



NOTE
Remember that **startstone** *uses (1) a GEMSTONE_SYS_CONF*

environment variable or (2) $GEMSTONE/data/system.conf as the default system-wide configuration file. If you want it to use parameters from a different configuration file, be sure to specify that file with the **startstone -z** *option.*

9.9 Version Compatibility

It is not always possible to restore backups made by a previous version of GemStone into a new version. Since kernel classes and methods are also included in the full backup, restoring an older version will result in GemStone Smalltalk code that is not correct for the GemStone version.

If you archive backups of your GemStone repository over multiple upgrades of your GemStone installation, you should also archive the GemStone executables for each version.

9.10 Warm Standby Systems

GemStone's backup and restore mechanisms can be used to set up a secondary server, running almost in parallel with the primary server and ready to take over as quickly as possible in case of any failure of the primary system.

To do this, a backup of the primary server is restored into a separate location. This backup is left running in restore mode, and as the transaction logs are created on the primary server, they are immediately restored into the warm standby system.

In case of failure of the primary system, the warm standby replays the final transaction log, executes a commitRestore, and is immediately ready to use.

For details on how to set up a warm standby system, see "How To Operate a Duplicate Server/Warm Standby" on page 72.

Chapter

10 Managing Memory

Executing your application code will naturally require access to previously committed objects in the repository. These objects are faulted into the Gem session's memory to be examined or updated. When new objects are created, they must reside in memory while they are being modified.

GemStone automatically garbage-collects temporary objects that are no longer referenced, and clears out the space used by persistent, committed objects, when memory is needed. However, the memory available for any session is finite. If you need to create large temporaries or modify many objects within a transaction, you may need to tune your application to increase the available memory, or to use memory more efficiently.

This chapter discusses the following topics:

- **Memory Organization —** How a GemStone session's memory is organized.
- **Configuring Temporary Memory Usage —** How to configure temporary object memory, and debug out-of-memory problems.

10.1 Memory Organization

Each Gem session has a temporary object memory that is private to the Gem process and its corresponding session. This local object memory is divided into the following regions:

- `new` — Young temporary objects; includes two subspaces named `eden` and `survivor`
- `old` — Older temporary objects
- `pom` — Unmodified faulted-in objects, divided into ten subspaces
- `perm` — Faulted-in or created Classes and Metaclasses
- `code` — Instances of GsNMethod being executed, or recently executed
- `mE` — oopMap entries that map objId to in-memory objects for committed objects

Temporary objects are created in the `new` area of local object memory. When the `new` area fills up, a scavenge occurs which throws away unreferenced objects in `new`. After an object has survived a number of scavenges, it is copied to the `old` area. After the `old` area has grown by some amount or is almost full, a mark/sweep takes place, finding all live objects and then compacting the `new`, `old`, `perm`, and `code` areas as needed to remove dead objects.

Committed objects referenced by the session are copied from the shared page cache into the `pom`, `perm`, or `code` areas at the point they are first referenced by interpreter execution or a GCI call. (This is called a "copy-on-read" design.)

If a committed object in the `pom` area has been modified, it is copied to the `old` area if a scavenge occurs before the change is committed. Objects that are sent to the GCI client, such as GBS, are also moved to the `old` area, whether or not they are modified.

When the `pom` area becomes full, the contents of its oldest subspace (that is, the oldest 10%) are discarded, and that subspace is reused to continue faulting-in committed objects. Before the oldest subspace is recycled, any objects in the subspace that have been modified, or that are currently referenced from the interpreter stack, are copied to the `old` area.

At transaction commit, any committed objects that have been modified, and any new objects transitively reachable from those modified objects, are copied to new data pages in the shared cache. A transaction conflict check is then performed. If the commit succeeds, the in-memory state of all new objects copied to the shared cache is changed to "committed". The newly committed objects are now eligible to

be removed from temporary memory by a mark/sweep or scavenge if they are no longer directly referenced from temporary objects.

10.2 Configuring Temporary Memory Usage

You may encounter an OutOfMemory error if you create too large a graph of live temporary objects at any time, or if you try to modify too many committed objects in a single transaction. OutOfMemory is a fatal error that terminates the session.

Very large numbers of Classes can also fill up temporary object memory. Any class that is referenced by message send or iteration is loaded into the `perm` area, and its method dictionaries, classHistory, and so on are loaded into the `old` area.

Persistent objects that are in the export set for a GCI client, such as GemBuilder for Smalltalk, are also moved to the `old` area. This includes objects that are replicated but not modified.

If you find that your application is running out of temporary memory, you can use several GemStone environment variables to help you identify which parts of your application are triggering garbage collection. Once you've done that, you can set GemStone configuration options to provide the needed memory.

Configuration Options

The values for these options are set when the gem or topaz -l process is initialized. You cannot change these values without restarting the VM. For more about these options, see the descriptions that begin on page 369.

GEM_TEMPOBJ_CACHE_SIZE
: The maximum size (in KB) of temporary object memory. (This limit also applies to linked Topaz sessions and linked GemBuilder applications.) When you only change this setting, and the other GEM_TEMPOBJ... configuration options use default values, then all of the various spaces remain in proportion to each other.

GEM_TEMPOBJ_MESPACE_SIZE
: The maximum size (in KB) of the mE (Map Entries) space within temporary object memory. Unless you are trying to minimize the memory footprint on HP-UX or AIX, you should always leave GEM_TEMPOBJ_MESPACE_SIZE at its default value so that the system can calculate the appropriate value. Otherwise, you are at risk of premature OutOfMemory errors.

GEM_TEMPOBJ_OOPMAP_SIZE
: The size of the hash table (that is, the number of 8-byte entries) in the objId-to-

object map within temporary object memory. This option should normally be left at its default setting.

GEM_TEMPOBJ_POMGEN_SIZE
The maximum size (in KB) of the POM generation area in temporary object memory. The POM generation area holds unmodified copies of committed objects that have been faulted into a Gem, and is divided into ten subspaces. This option should normally be left at its default setting so that the system can calculate the value.

Methods for Computing Temporary Object Space

To find out how much space is left in the `old` area of temporary memory, the following methods in class System (category Performance Monitoring) are provided:

`System _tempObjSpaceUsed`
Returns the approximate number of bytes of temporary object memory being used to store objects.

`System _tempObjSpaceMax`
Returns the approximate maximum number of bytes of temporary object memory that are usable for storing objects.

`System _tempObjSpacePercentUsed`
Returns the approximate percentage of temporary object memory that is in use to store temporary objects. This is equivalent to the expression:

```
(System _tempObjSpaceUsed * 100) //
  System _tempObjSpaceMax.
```

Note that it is possible for the result to be slightly greater than 100%. Such a result indicates that temporary memory is almost completely full.

Sample Configurations

This section presents several sample configurations:

- Default configuration
- Larger old area, smaller pom
- Smaller old area, larger pom

These examples assume that you have already set the GS_DEBUG_VMGC... environment variables (page 297) to produce the resulting printouts. The examples shown are for Solaris and may vary on other platforms.

Default Configuration

A default value (10000) for GEM_TEMPOBJ_CACHE_SIZE (that is, 10 MB) produces a limit of about 7 MB of temporary plus modified-committed objects (`old`), space for a working set of about 8 MB of unmodified committed objects (`pom`), and a maximum memory footprint on the order of 25 MB.

The following example shows the printout for this configuration:

```
vmGc spaceSizes: eden init 1864K max 1864K, survivor init
320K max 320K,
vmGc old max 7496K, code max 2000K, perm max 1000K, pom
10 * 840K=8400K,
vmGc remSet 216K, meSpace max 9592K oopMapSize 65536
```

(The internal structures `remSet`, `meSpace`, and `oopMapSize` are not of interest here.)

Larger old, Smaller pom

The following settings configure the application for a 5 MB working set of unmodified committed objects (smaller than the default), and a maximum of 25 MB of temporary plus modified objects (larger than the default).

```
GEM_TEMPOBJ_CACHE_SIZE = 25000;
GEM_TEMPOBJ_POMGEN_SIZE = 5000;
```

The following example shows the printout for this configuration:

```
vmGc spaceSizes: eden init 2000K max 4688K, survivor init
400K max 784K
vmGc old max 18744K, code max 5000K, perm max 2504K, pom
10 * 504K=5040K
vmGc remSet 360K, meSpace max 16824K oopMapSize 65536
```

Smaller old Area, Larger pom

The following settings configure an application with a large working set of committed objects and small temporary object space.

```
GEM_TEMPOBJ_CACHE_SIZE = 7000;
GEM_TEMPOBJ_POMGEN_SIZE = 100000;
```

The following example shows the printout for this configuration:

```
vmGc spaceSizes: eden init 1304K max 1304K , survivor
init 224K max 224K
vmGc old max 5248K, code max 1400K, perm max 704K, pom 10
* 10000K=100000K
vmGc remSet 1008K, meSpace max 48880K oopMapSize 524288
```

Debugging out-of-memory errors

When any of the following environment variables are set to a positive non-zero value, they have the effect described here for each Gem or linkable Topaz (topaz -l) process that you subsequently start. For all of these environment variables, the printout goes to the "output push" file of a linkable Topaz (topaz -l) session, for use in testing your application. If that file is not defined, the printouts go to standard output of the session's Gem or topaz -l process.

> *To set any of these environment variables, you must first uncomment them in the* `$GEMSTONE/sys/gemnetdebug` *file. Be aware that the contents of* `gemnetdebug` *are subject to change at any time. For the most current information about these and other variables, examine the* `gemnetdebug` *file.*

GS_DEBUG_COMPILE_TRACE
: Trace method compiles. The following are valid values:
  0 - no tracing
  1 - one line (class,selector) of each method compiled
  2 - in addition to above, bytecode disassembly
  3 - in addition to above, native code assembly listing

GS_DEBUG_VMGC_MKSW_MEMORY_USED_SOFT_BREAK
: At the end of each mark/sweep, if the percent of memory used is greater than the threshold specified by this variable, a SoftBreak (error 6003) is generated, and the threshold is raised by 5 percent. We suggest a setting of 75%.

GS_DEBUG_VMGC_MKSW_PRINT_STACK
: The mark/sweep count at which to begin printing the Smalltalk stack at each mark/sweep.

GS_DEBUG_VMGC_MKSW_PRINT_C_STACK
: The mark/sweep count at which to begin printing the C stack at each mark/sweep. This variable is very expensive, consuming two seconds plus the cost of fork() for each printout.

GS_DEBUG_VMGC_PRINT_MKSW
The mark/sweep count at which to begin printing mark/sweeps.

GS_DEBUG_VMGC_PRINT_MKSW_MEMORY
The mark/sweep count at which to begin printing detailed memory usage (20 lines) for each mark/sweep.

GS_DEBUG_VMGC_PRINT_MKSW_MEMORY_USED
Specifies when Smalltalk stack printing starts as the application approaches OutOfMemory conditions. At the end of each mark/sweep, if the percent of memory used is greater than the threshold specified by this variable, the mark/sweep is printed, the Smalltalk stack is printed, and the threshold is raised by 5 percent. In a situation producing an OutOfMemory error, you should get several Smalltalk stacks printed in the Gem log file before the session dies.

GS_DEBUG_VMGC_PRINT_SCAV
The scavenge count at which to begin printing scavenges. Once this takes effect, all mark/sweeps will also be printed. Be aware that printing scavenges can produce large quantities of output.

GS_DEBUG_VM_PRINT_TRANS
Print transaction boundaries (begin/commit/abort) in the log file.

GS_DEBUG_VMGC_SCAV_PRINT_STACK
The scavenge count at which to begin printing the Smalltalk stack at each scavenge. Be aware that this print activity can produce large quantities of output.

GS_DEBUG_VMGC_SCAV_PRINT_C_STACK
The scavenge count at which to begin printing the C stack at each scavenge. This variable is very expensive, consuming 2 seconds plus the cost of fork() for each printout.

GS_DEBUG_VMGC_VERBOSE_OUTOFMEM
Automatically call the primitive for `System class>>_vmPrintInstanceCounts:0` when an OutOfMemory error occurs, and also print the Smalltalk stack. (For details about this method, see the comments in the image.) This applies to each Gem or linkable Topaz (topaz -l) process that you subsequently start.

GS_DEBUG_VMGC_VERIFY_MKSW
The mark/sweep count at which to begin verifying object memory before and after each mark/sweep.

GS_DEBUG_VMGC_VERIFY_SCAV

The scavenge count at which to begin verifying object memory before and after each scavenge. Once this takes effect, GS_DEBUG_VMGC_VERIFY_MKSW will also be in effect. Be aware that this activity uses significant amounts of CPU time.

Platform consideration on memory allocation

On Solaris and Linux, the memory spaces configured by GEM_TEMPOBJ_CACHE_SIZE and GEM_TEMPOBJ_POMGEN_SIZE are reserved in virtual memory at Gem startup, and are allocated/released as needed via mmap() calls to Solaris or Linux. On HP-UX and AIX, memory is handled differently, and may not be reserved without being allocated. On these platforms, the memory configured by GEM_TEMPOBJ_CACHE_SIZE must all be allocated at Gem startup.

Signal on low memory condition

When a session runs low on temporary object memory, there are actions it can take to avoid running out of memory altogether. By enabling handling for the signal `AlmostOutOfMemory`, an application can take appropriate action before memory is entirely full. This signal is asynchronous, so may be received at any time memory use is greater than the threshold the end of an in-memory markSweep. However, if the session is executing a user action, or is in index maintenance, the error is deferred and generated when execution returns.

When performing index operations, such as creating indexes for large collections, the IndexManager can be configured to use this facility to automatically commit when memory is low. Committing objects allows them to be removed from memory, since they can be re-loaded as needed from the persistent object. See the *Programming Guide for GemStone/S 64 Bit*, chapter 5, for details on IndexManager autoCommit.

For more information on handling `AlmostOutOfMemory`, see the *Programming Guide for GemStone/S 64 Bit*.

Chapter

11 Managing Growth

In the course of everyday operations, your GemStone/S 64 Bit repository will grow. Some of this growth will be the result of new data in your repository, but some will represent unreferenced or outdated objects. These objects, no longer needed, must be removed to prevent the repository from growing arbitrarily large. The process of removing unwanted objects to reclaim their storage is referred to as *garbage collection*.

This chapter describes GemStone's garbage collection mechanisms and explains how and when to use them.

This chapter discusses the following topics:

- **Basic Concepts —** The main concepts underlying garbage collection (page 304).
- **Garbage Collection Operations —** markForCollection (page 313), FDC/MGC (page 316), epoch garbage collection (page 319), and reclaim (page 328).
- **GcGems —** How to configure, start, and stop Admin GcGems (page 331) and Reclaim GcGems (page 333).
- **Tuning Garbage Collection —** Tuning multi-threaded scan operations, and other special issues affecting Garbage Collection (page 339).

11.1 Basic Concepts

Smalltalk execution can produce a number of objects needed only for the moment. In addition, normal business operations can cause previously committed objects to become obsolete. To make the best use of system resources, it is desirable to reclaim the resources these objects use as soon as possible.

Different Types of Garbage

Garbage collection mechanisms vary according to *where* garbage collection occurs — temporary (scratch) memory or permanent object space — and *how* it occurs — automatically, or in response to an administrator's action.

Each Gem session has its own private memory intended for scratch space, known as *local object memory*. The Gem session uses local object memory for a variety of temporary objects, which can be garbage-collected individually. This type of garbage collection is handled automatically by the session and is (for the most part) not configurable, although memory can be configured for specific gem requirements. These issues are covered in Chapter 10, "Managing Memory" on page 295.

Permanent objects are organized in units of 16 KB called *pages.* Pages exist in the Gem's private page cache, the Stone repository monitor's private page cache, the shared page cache, and on disk in the extents. When first created, each page is associated with a specific transaction; after its transaction has completed, GemStone does not write to that page again until all its storage can be reclaimed.

Objects on pages are not garbage-collected individually. Instead, the presence of a shadow object or dead object triggers reclamation of the page on which the object resides. Live objects on this page are copied to another page.

The Process of Garbage Collection

Removing unwanted objects is a two-phase process:

1. Identify—*mark*—superfluous objects.
2. *Reclaim* the resources they consume.

Together, *marking* and *reclaiming* unwanted objects is *collecting garbage.*

Complications ensue because each Gem in a transaction is guaranteed a consistent view of the repository: all visible objects are guaranteed to remain in the same state as when the transaction began. If another Gem commits a change to a mutually visible object, both states of the object must somehow coexist until the older transaction commits or aborts, refreshing its view. Therefore, resources can be

reclaimed only after all transactions concurrent with marking have committed or aborted.

Older views of committed, modified objects are called *shadow objects.*

Garbage collection reclaims three kinds of resources:

- The storage occupied by dead objects
- The storage occupied by shadow objects
- Object identifiers (OOPs) for dead objects

Live objects

GemStone considers an object *live* if it can be reached by traversing a path from AllUsers, the root object of the GemStone repository. By definition, AllUsers contains a reference to each user's UserProfile. Each UserProfile contains a reference to the symbol list for a given user, and those symbol dictionaries in these lists in turn point to classes and instances created by that user's applications. Thus, AllUsers is the root node of a tree whose branches and leaves encompass all the objects that the repository requires at a given time to function as expected.

Transitive closure

Traversing such a path from a root object to all its branches and leaves is called *transitive closure.*

Dead objects

An object is *dead* if it cannot be reached from the AllUsers root object. Other dead objects may refer to it, but no live object does. Without living references, the object is visible only to the system, and is a candidate for reclamation of both its storage and its OOP.

Shadow objects

A *shadow object* is a committed object with an outdated value. A committed object becomes shadowed when it is modified during a transaction. Unlike a dead object, a shadow object is still referenced in the repository because the old and new values share a single object identifier.The shadow object must be maintained as long as it is visible to other transactions on the system; then the system can reclaim only its storage, not its OOP (which is still in use identifying the committed object with its current value).

Commit records

Views of the repository are based on *commit records,* structures written when a transaction is committed. Commit records detail every object modified *(the write set),* as well as the new values of modified objects. The Stone maintains these commit records; when a Gem begins a transaction or refreshes its view of the repository, its view is based on the most recent commit record available.

Each session's view is based on exactly one commit record at a time, but any number of sessions' views can be based on the same commit record.

NOTE

The repository must retain each commit record and the shadow objects to which it refers as long as that commit record defines the transaction view of any session.

Commit record backlog

The list of commit records that the Stone maintains in order to support multiple repository views is the *commit record backlog.*

Shadow or Dead?

The following example illustrates the difference between dead and shadow objects. In Figure 11.1, a user creates a SymbolAssociation in the SymbolDictionary Published. The SymbolAssociation is an object (oop 27111425) that refers to two other objects, its instance variables key (#City, oop 20945153), and value ('Beaverton', oop 27110657).

The Topaz command "display oops" causes Topaz to display within brackets ([ ]) the identifier, size, and class of each object. This display is helpful in examining the initial SymbolAssociation and the changes that occur.

Figure 11.1 An Association Is Created and Committed

```
topaz 1> display oops
topaz 1> printit
Published at: #City put: 'Beaverton'.
Published associationAt: #City
%
[27111425 sz:2 cls: 111617 SymbolAssociation] a SymbolAssociation
  key                  [20945153 sz:4 cls: 110849 Symbol] City
  value                [27110657 sz:9 cls: 74753 String] Beaverton
topaz 1> commit
Successful commit
```

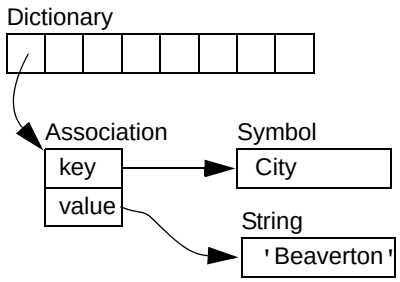


Figure 11.2 shows a second Topaz session that logs in at this point. Notice that the Topaz prompt identifies the session by displaying a digit. Because Session 1 committed the SymbolAssociation to the repository, Session 2 can see the SymbolAssociation.

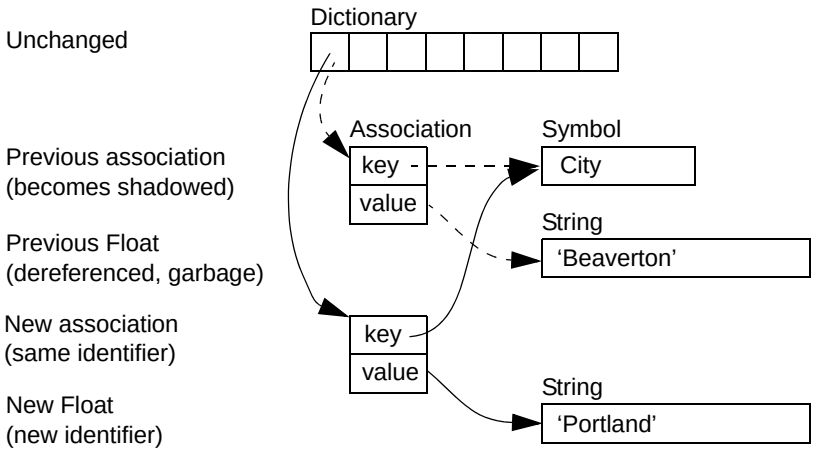
Figure 11.2 A Second Session Can See the Association

```
topaz 2> display oops
topaz 2> printit
Published associationAt: #City .
%
[27111425 sz:2 cls: 111617 SymbolAssociation] a SymbolAssociation
  key                  [20945153 sz:4 cls: 110849 Symbol] City
  value                [27110657 sz:9 cls: 74753 String] Beaverton
```

Now Session 1 changes the *value* instance variable, creating a new SymbolAssociation (Figure 11.3). Notice in the oops display that the new SymbolAssociation object has the same identifier (27111425) as the previous Association.

Figure 11.3 The Value Is Replaced, Changing the Association

```
topaz 1> printit
City := 'Portland'.
Published associationAt: #City.
%
[27111425 sz:2 cls: 111617 SymbolAssociation] a SymbolAssociation
  key                  [20945153 sz:4 cls: 110849 Symbol] City
  value                [27109121 sz:8 cls: 74753 String] Portland
topaz 1> commit
Successful commit
```



- The SymbolAssociation is now *shadowed*. Because the shadow SymbolAssociation was part of the committed repository and is still visible to other transactions (such as that of Session 2), it cannot be overwritten. Instead, the new SymbolAssociation is written to another page, one allocated for the current transaction.
- The previous value (oop 27110657) is no longer referenced in the repository. For now, this object is considered *possibly dead;* we cannot be sure it is dead because, although the object has been dereferenced by a committed transaction, other, concurrent transactions might have created a reference to it.

Even though Session 1 committed the change, Session 2 continues to see the original SymbolAssociation and its value (Figure 11.4). Session 2 (and any other concurrent sessions) will not see the new SymbolAssociation and value until it either commits or aborts the transaction that was ongoing when Session 1 committed the change.

Figure 11.4 Session 2 Sees Change After Renewing Transaction View of Repository

```
topaz 2> printit
Published associationAt: #City.
%
[27111425 sz:2 cls: 111617 SymbolAssociation] a SymbolAssociation
  key                 [20945153 sz:4 cls: 110849 Symbol] City
  value               [27110657 sz:9 cls: 74753 String] Beaverton

topaz 2> abort
topaz 2> printit
Published associationAt: #City.
%
[27111425 sz:2 cls: 111617 SymbolAssociation] a SymbolAssociation
  key                 [20945153 sz:4 cls: 110849 Symbol] City
  value               [27109121 sz:8 cls: 74753 String] Portland
```

Only when all sessions with concurrent transactions have committed or aborted can the shadow object be garbage collected.

What Happens to Garbage?

This section describes the steps involved in garbage collection. Specific garbage collection mechanisms will follow these steps, although the details will vary when using different garbage collection mechanisms.

The basic garbage collection process encompasses nine steps:

1. Find all the live objects in the system by traversing references, starting at the system root AllUsers. This step is called *mark/sweep*.
2. The Gem that performed mark/sweep now has a list of all live objects. It also knows the universe of all possible objects: objects whose OOPs range from zero to the highest OOP in the system. It can now compute the *set of possible dead objects* as follows:

   a. Subtract the live objects from the universe of possible objects.

   b. Subtract all the unassigned (free) OOPs in that range.

This step is called the *object table sweep* because the Gem uses the object table to determine the universe of possible objects and the unassigned OOPs.

3. The Gem performing this work now has a list of *possibly dead* objects. We can't be sure they're dead because, during the time that the mark/sweep and object table sweep were occurring, other concurrent transactions might have created references to some of them.

   The Gem sends the Stone the *possible dead set* and returns.

4. Now, in a step called *voting,* each Gem logged into the system must search its private memory to see if it has created any references to objects in the possible dead set. When it next commits or aborts, it votes on every object in the possible dead set. Objects referenced by a Gem are removed from the possible dead set.

   NOTE

   Gems do not vote until they complete their current transaction. If a Gem is sleeping or otherwise engaged in a long transaction, the vote cannot be finalized and garbage collection pauses at this point. Commit records accumulate, garbage accumulates, and a variety of problems can ensue.

5. Because all the previous steps take time, it's possible that some Gems were on the system when the mark/sweep began, created a reference to an object now in the possible dead set, and then logged out. They cannot vote on the possible dead set, but objects they've modified are in the write sets of their commit records. The *Admin GcGem,* a process dedicated to administrative garbage collection tasks, scans all these write sets (the *write set union),* and votes on their behalf. This is called the *write set union sweep.*

6. After all voting is complete, the resulting set now holds nothing but unreferenced objects. The Stone now promotes the objects from possibly dead to dead.

7. One or more *Reclaim GcGems* reclaims pages: it copies live objects on the page onto a new page, thereby compacting live objects in page space. The page now contains only recycleable objects and perhaps free space.

8. The Reclaim GcGem commits. The reclaimed OOPs are returned to their free pool.

9. The Reclaim GcGem's commit record is disposed of. The reclaimed pages are returned to their free pool.

Admin and Reclaim GcGems

It is useful to understand the distinction between the Admin GcGem and Reclaim GcGems:

- The Admin GcGem finalizes the vote on possibly dead objects (page 310, Step 5), and performs the write set union sweep. The Admin GcGem also performs epoch garbage collection (page 313), if enabled.

  There is only one Admin GcGem running (or none).

  The Admin GcGem is started by the Stone at startup time if enabled by the STN_ADMIN_GC_SESSION_ENABLED configuration option (page 332).

- Reclaim GcGems are dedicated to the task of reclaiming shadowed pages and dead objects repository-wide, along with their OOPs.

  There may be multiple Reclaim GcGems running (or none). If you have multiple extents in your repository, you can run up to one Reclaim GcGem for each extent file in the repository. Each Reclaim GcGem can handle several extents.

  The configuration parameter STN_NUM_GC_RECLAIM_SESSIONS specifies the maximum number of Reclaim GcGems that should be started (page 334).

By default, one Admin GcGem and one Reclaim GcGem are configured to run, and are started automatically when the Stone is started. This is the simplest GcGem configuration. Also by default, epoch is disabled.

- We recommend that you leave the Admin GcGem running at all times, although this GcGem is required only following a `markForCollection` or `markGcCandidatesFromFile:`, or epoch garbage collection operation. (Subsequent sections of this chapter describe these operations in detail.) If the Admin GcGem is not running following one of these operations, the garbage collection process cannot complete, and garbage can build up in your system.
- We recommend that you have at least one Reclaim GcGem running at all times, to reclaim shadow objects.

Both the Admin and Reclaim GcGems are run from the GcUser account, a special account that logs in to the repository to perform garbage collection tasks. For more about this account, see Chapter 6, “User Accounts and Security“.

GemStone’s Garbage Collection Mechanisms

GemStone provides the following mechanisms that together mark and reclaim garbage, thereby helping you to control repository growth.

Marking

Repository-wide marking — To prevent the repository from growing large enough to cause problems on a regular basis, you can run `Repository >> markForCollection` (page 313). This method combines a full sweep of all objects in the repository and the marking of each possible dead object in a single operation.

FDC/MGC — This two-step process allows the first step to be performed on an offline copy of the repository, reducing the impact on the production system. In the first step (FDC), the method `Repository >> findDisconnectedObjectsAndWriteToFile:` explicitly finds disconnected objects in the repository and identifies a list of possible targets for garbage collection, and writes the possibly dead objects to a file. In the second step (MGC), the method `Repository >> markGcCandidatesFromFile:` examines the set of candidate garbage objects, ensures that they are not referenced by live objects, and then marks them as dead for subsequent garbage collection. For details about the FDC/MGC process, see page 316.

Epoch garbage collection — If enabled, the Admin GcGem periodically examines all transactions written since a specific, recent time (the beginning of this *epoch*) for objects that were created and then dereferenced during that period. However, epoch garbage collection cannot reclaim objects that are created in one epoch but dereferenced in another. In spite of its name, epoch garbage collection only marks; it does not reclaim. You can configure various aspects to maximize its usefulness. Epoch garbage collection is disabled by default. For details about epoch garbage collection, see page 313.

Reclaiming

Reclaim — Once you've run `markForCollection`, FDC/MGC, or epoch garbage collection, you can run Reclaim GcGems to reclaim pages that contain either dead or shadow objects. When there are a high number of objects needing to be reclaimed, you can run multiple Reclaim GcGems simultaneously, as many as one Reclaim GcGem for each extent in your repository. For details about reclaiming pages, see page 333.

GcLock

Many garbage collection process, such as mark/sweep operations, should not be run concurrently. To prevent this, there is a shared internal lock called the GcLock. Garbage collection processes that cannot run concurrently, such as

markForCollection, get the GcLock, which prevents another one from starting up. The GcLock is also held by the Admin GcGem at certain periods.

In addition to garbage collection, some repository-wide operations such as GsObjectInventory (page 171) also hold the GcLock while they are running.

If another task that requires the GcLock is in progress at the time you try to do `markForCollection` or `findDisconnectedObjects...`, they will not execute, but report an error similar to that shown below.

```
-- Request for MFC gclock by session 6 denied, reason: vote
state is voting, waiting for promote dead
-------------------------------------------------
GemStone: Error         Nonfatal
a Error occurred (error 2501), An operation that needs to
acquire the gcLock was not able to get the lock within a
reasonable amount of time.
Error Category: 231169 [GemStone] Number: 2501  Arg Count: 1
Context : 27095553 exception : 27097089
Arg 1: [27097345 sz:98 cls: 74753 String] Request for MFC
gclock by session 6 denied, reason: vote state is voting,
waiting for promote dead
```

The cause of the conflict may be:

- Another operation that requires the GcLock is in progress in another session; this includes epoch, MFC, and MGC, and also operations such as GsObjectInventory.
- A previous epoch, `markGcCandidatesFromFile:` or `markForCollection` completed the mark phase, but voting on possibly dead objects has not completed.

  For voting to complete, the Admin GcGem must be running, as described on page 331. Also, any long-running session that neither aborts nor commits will prevent the vote from completing.

11.2 MarkForCollection

Privileges required: GarbageCollection.

The method `Repository>>markForCollection` sweeps the entire repository and marks as live all objects that can be reached through a transitive closure on the symbol lists in AllUsers, as described on page 305. The remaining objects become the list of possible dead objects.

markForCollection only provides a set of possible dead objects for voting and eventual reclaiming as described under "What Happens to Garbage?" on page 309. It does not reclaim the space or OOPs itself—the Reclaim GcGems do that, as described beginning on page 333.

To mark unreferenced GemStone objects for collection, log in to GemStone and send your repository the message markForCollection, as in the following example:

```
topaz 1> printit
SystemRepository markForCollection
%
```

If you are performing markForCollection on a large production repository, consider the steps described under "Impact on Other Sessions" on page 315.

When markForCollection completes successfully, the Gem that started it displays a message such as the one below:

```
a Notification occurred (notification 3020), Successful
completion of markForCollection.
    2694458 live objects found.
    129654 possible dead objects, occupying approximately
11668860 bytes, may be reclaimed.  (error 3020)
```

This method aborts the current transaction and runs markForCollection inside a transaction, but monitoring the commit record backlog so it can abort as necessary to prevent the backlog from growing. When markForCollection completes, the session reenters a transaction, if it was in one when this method was invoked.

If another garbage collection task is in progress at the time you try to do markForCollection, this method will retry for a fixed period, reporting status. If the other operation does not complete within the timeout period, it reports an error indicating it could not get the GcLock. See "GcLock" on page 312 for more details.

Before issuing the error, the markForCollection method waits up to a minute for the other operation to complete. To have the markForCollection wait for a longer period, use markForCollectionWait: *waitTimeSeconds*. To wait as long as necessary for the other garbage collection to finish, pass the argument -1. Do so with caution, however; under certain conditions, the session could wait forever. To avoid this:

- Make sure that other sessions are committing or aborting, which allows voting on possible dead to complete.

- Make sure that the Admin GcGem is running to complete processing of dead objects once the vote is completed.

Impact on Other Sessions

The `markForCollection` operation uses multi-threaded scan. For more details on this, see "Multi-Threaded Scan" on page 339.

By default, `markForCollection` uses up to 90% of the CPUs and up to two threads, which is a medium-aggressive load.

To enable `markForCollection` to complete as quickly as possible, you can use:

```
SystemRepository fastMarkForCollection
```

This uses higher settings (95% of CPU, and a number of threads based on the current hardware) to use as many system resources as possible. The performance of anything else running on the same system may be heavily degraded.

For maximum control, use the method

```
SystemRepository markForCollectionWithMaxThreads: threadsCt
   waitForLock: seconds
   percentCpuActiveLimit: percentLimit
```

This allows you to specify the precise limits.

Additional impact on other sessions may occur after the `markForCollection` has completed because the results may cause an increase in the number of pages that are candidates for reclaiming.

Scheduling markForCollection

To invoke `markForCollection` using the **`cron`** facility, create a three-line script file similar to the Topaz example on page 314 by entering everything except the prompt. Use this script as standard input to **`topaz`**, and redirect the standard output to another file:

`topaz <` *scriptName* **`>`** *logName*

Make sure that $GEMSTONE and any other required environment variables are defined during the **`cron`** job. Either create a `.topazini` file for a user who has GarbageCollection privilege, or insert those login settings at the beginning of the script. For information about using **`cron`**, refer to your operating system documentation.

11.3 The FDC/MGC Process

As discussed in the previous section, `markForCollection` combines a full sweep of all objects in the repository and marking of each possible dead object in a single operation. In a large production system, this can have a significant impact on performance. To lessen this impact, you can perform garbage collection as a two-step process: `findDisconnectedObjectsAndWriteToFile:` (FDC) and `markGcCandidatesFromFile:` (MGC). These two steps together perform the same function as running `markForCollection`.

In this two-step process, the possibly dead objects are first written to a file, thus allowing you to run the FDC in a copy of the repository rather than in the production system. The OOPs of the possibly dead objects are then transferred to the production system, where they are marked as dead and subsequently garbage-collected.

Running in a copy of the repository avoids the performance impact of this processing on the production repository, and permits operations on the repository that would not be possible during a long-running repository sweep; for example, the production repository can be shut down.

> *If you prefer to run garbage collection in the production system rather than in a copy of the repository,* `markForCollection` *would be a better option.*

For offline FDC/MGC to be effective, Epoch garbage collection must be disabled on the production repository during the period from the time the production copy is made, until the markGcCandidates completes. If the possibly dead objects that are passed into the MGC operation contain objects that have already been garbage collected and reused, performance degrades to the point of uselessness. Epoch garbage collection is disabled by default.

Step 1. Prepare for FDC.

FDC produces a large file of encoded OOPs; the size of the file (in bytes) is slightly more than eight times the number of dead objects. On the server that will run the backup, ensure that there is enough space for the resulting file, and that the filename you will use does not already exist.

Step 2. Create a copy of the production repository.

Make a backup of your production repository. You may either make an online extent file backup (page 259), a Smalltalk full backup (page 264), or an offline extent file backup (page 286).

Restore the backup into another location. This is the repository that will run the FDC, while your production repository continues to operate normally.

Step 3. Run `findDisconnectedObjectsAndWriteToFile:...`.

On the running backup repository, execute the FDC method:

```
SystemRepository
  findDisconnectedObjectsAndWriteToFile: filename.txt
  pageBufferSize: 128
  saveToRepository: true
```

This method does the following:

- Runs code similar to the `markForCollection` sweep phase on the backup repository.
- Stores the resulting list of candidate dead objects to the named file (*filename.txt*), as encoded OOPs. If a relative filename is specified, the file is created relative to the current working directory. Note that for RPC sessions the working directory is the home directory of the gem process' UNIX user.
- If `saveToRepository:` is true, stores the list of encoded OOPs to an internal array as well. Set this argument to true if you want to be able to subsequently examine and analyze the results of the FDC operation; it is not needed as part of the garbage collection process.

If `saveToRepository:` is true, you can subsequently use the array of dead objects to recreate the FDC file in the event that the file is deleted or lost. See the following discussion, "If You Need to Recreate the FDC File."

Also note that a larger `pageBufferSize` may improve performance, provided you have adequate memory. See "Memory Impact" on page 341.

Step 4. At this point, you can shut down the backup repository; it is no longer needed for the MGC/FDC process.

Step 5. Run `markGcCandidatesFromFile:` (page 318) on the production system.

If You Need to Recreate the FDC File

If your FDC file should inadvertently be deleted or become lost, you may still be able to recreate the FDC file for subsequent use by `markGcCandidatesFromFile:` (MGC) — provided that `saveToRepository:` was true when you ran FDC.

You can execute this method to write the array of candidate dead objects found by the last FDC operation (for which `saveToRepository:` was true) to a file:

```
SystemRepository writeFdcArrayToFile: aFileNameString
```

NOTE

Before running this method, ensure that you have sufficient disk space to hold the entire file, whose size in bytes will be approximately four times the size of the array.

This method expects Globals at: #FdcResults to contain the array of candidate dead objects, in order and sorted by OOP. You must specify the file as a String, using the full path and filename of a file that does not yet exist.

The method returns true if successful, false if the entire file could not be written, or nil if the results of the last FDC could not be found.

Run markGcCandidatesFromFile:

Privileges required: GarbageCollection.

`markGcCandidatesFromFile:` (MGC) performs an optimized linear sweep of the repository to identify any objects in the FDC output file that still have references. If the only references to candidate objects are from other candidates, the objects are marked for subsequent reclamation.

The MGC operation scans all *non-candidate* objects to ensure that there are no references to candidate objects. If there are no such references, the MGC operation is complete. If, however, there are any references to candidate objects, the MGC operation performs a transitive closure on all *former candidate* objects to identify any other candidate objects that cannot be collected.

To minimize the impact on performance, run the MGC operation at a time when the load on the production repository is low, such as during off-hours.

On the production system, execute:

```
SystemRepository markGcCandidatesFromFile: aFileNameString
```

where *aFileNameString* is a String representing the full path to the file of encoded OOPs that was created during the FDC (page 317, Step 3). Make sure that this file is available from the production location.

markGcCandidates requires the GcLock. If another garbage collection (`markForCollection`) is in progress at the time you try to do `markGcCandidatesFromFile:`, or if voting has not completed, it may report a

conflict error similar to that shown on page 313. Try `markGcCandidatesFromFile:` again after a few minutes.

Recall that all marking only provides a set of possible dead objects for voting and eventual reclaiming, as described under "What Happens to Garbage?" on page 309. Marking does not by itself reclaim space or identifiers—the Reclaim GcGems do that, as described beginning on page 333.

As mentioned elsewhere in this chapter, the Admin GcGem must be running to complete processing of dead objects following an MGC operation. For details, see "Running the Admin GcGem" on page 331.

In some cases, there could be irrelevant errors that would prevent MGC from completing. To override such errors, execute the following method:

```
SystemRepository markGcCandidatesFromFile: aFileNameString
  forceOnError: aBoolean
```

This method is virtually identical to `markGcCandidatesFromFile:`, except that if *aBoolean* is true, the MGC will run even if the given file contains objects that fail validation, or there are references from live objects to some objects in the candidate collection. The live objects in the candidate collection are not marked as dead.

Impact on Other Sessions

The FDC operation uses multi-threaded scan. For more details on this, see "Multi-Threaded Scan" on page 339.

Since FDC is designed to be run in an offline copy of the repository, by default, FDC uses very aggressive settings, 95% of CPU and two threads per CPU core. It may consume all CPU and disk I/O host system resources.

If you want to reduce the impact of the FDC, you can use the basic method

```
basicFindDisconnectedObjectsAndWriteToFile: aFileNameString
 pageBufferSize: pageBufSize saveToRepository: saveToRep
 withMaxThreads: maxThreads maxCpuUsage: aPercentage
```

which allows you to include the specific thread and CPU limits.

11.4 Epoch Garbage Collection

Privileges required: GarbageCollection.

Epoch garbage collection operates on a finite set of recent transactions: the *epoch.* Using the write set that the Stone maintains for each transaction, the Admin

GcGem examines every object created during the epoch. If an object is unreferenced by the end of the epoch, it is marked as garbage and added to the list of possible dead objects.

Epoch collection is efficient because:

- It's faster and easier to perform a transitive closure on a few recent transactions than on the entire repository.
- Most objects die young, especially in applications characterized by numerous small transactions updating a few previously committed objects. An epoch of the right length can collect most garbage automatically.

Although epoch collection identifies a lot of dead objects, it cannot replace `markForCollection` because it will never detect objects created in one epoch and dereferenced in another.

By default, epoch garbage collection is disabled. You can enable it in either of two ways:

- Before you start the Stone, set the STN_EPOCH_GC_ENABLED configuration parameter to TRUE. For details about this parameter, see page 378.
- Execute the method `System class >> enableEpochGc`. You may also manually disable epoch garbage collection using `System class >> disableEpochGc`. Using these methods updates the system configuration file.

After your installation has been operating for a while, and you've had the chance to collect operational statistics, consider this: epochs of the wrong length can be notably inefficient. An in-depth discussion of the performance trade-offs of short or long epochs starts on page 322.

Running Epoch Garbage Collection

When epoch garbage collection is enabled, it will run automatically according to the GcUser configuration parameters #epochGcTimeLimit and #epochGcTransLimit.

You can force an epoch garbage collection to begin using `System class >> forceEpochGc`.

`forceEpochGc` will return false, and not start an epoch garbage collection, if any of the following are true:

- Checkpoints are suspended.
- Another garbage collection operation is in progress.

- Unfinalized possible dead objects exist (that is, `System voteState` returns a non-zero value).
- The system is in restore mode.
- The Admin GcGem is not running.
- Epoch garbage collection is disabled (that is, STN_EPOCH_GC_ENABLED = FALSE).
- The system is performing a reclaimAll.
- A previous `forceEpochGc` operation was performed and the epoch has not yet started or completed.

Tuning Epoch

Epoch Configuration Parameters

The following configuration parameters are available to control the performance of epoch garbage collection, and are stored in GcUser's UserGlobals. For an example of how to change these parameters, see page 342.

`#epochGcTimeLimit` The maximum frequency of epoch garbage collection (in seconds). Default: one hour (3600 seconds). This value should be at least 1800 (30 minutes), since the aging of objects faulted into Gem memory uses 5 minute aging for each of 10 subspaces of the POM generation.

`#epochGcTransLimit` The minimum number of transactions required to trigger epoch garbage collection. Default: 5000.

`#epochGcPercentCpuActiveLimit`
Limit active epoch threads when system percentCpuActive is above this limit. Default: 90.

`#epochGcPageBufferSize`
Size in pages of buffer used for epoch GC (must be power of 2). Default: 64.

`#epochGcMaxThreads` The MaxThreads used for next epochGc.

Epoch garbage collection uses the multi-threaded scan (see page 339) and can be tuned to complete more quickly with more system performance and resources impact, or take longer and use fewer system resources. The parameters `#epochGcPercentCpuActiveLimit`, `#epochGcPageBufferSize`, and `#epochGcMaxThreads` are used to tune epoch garbage collection's multi-threaded scan impact.

Determining the Epoch Length

Epoch garbage collection's ability to identify unreferenced objects depends on the relationship between three variables:

- The rate of production *R* of short-lived objects.
- The lifetime *L* of these objects.
- The epoch length *E*.

The only variable under your direct control is epoch length. Although you cannot specify it explicitly, the following configuration parameters jointly control the length of an epoch:

- `#epochGcTimeLimit`
- `#epochGcTransLimit`

Epoch garbage collection occurs when:

```
(the time since last epoch > epochGcTimeLimit ) AND
      (transactions since last epoch > epochGcTransLimit)
```

The following discussion assumes that the epoch is determined by the minimum time interval (`#epochGcTimeLimit`) because other threshold is always met.

Figure 11.5 shows the effect of the epoch on the number of items marked. If L = E, for example, five minutes, every object's lifetime spans epochs (top part of graph), and none are collected.

When the epoch is longer than an average object's lifetime, however, some objects live and die within the same epoch, and can be marked. The lower part of Figure 11.5 shows an example where $E = 3L$ and objects are created at a uniform rate. Objects created during the first two-thirds of the interval die before its end and are marked. Only those created during the final third survive to the next epoch.

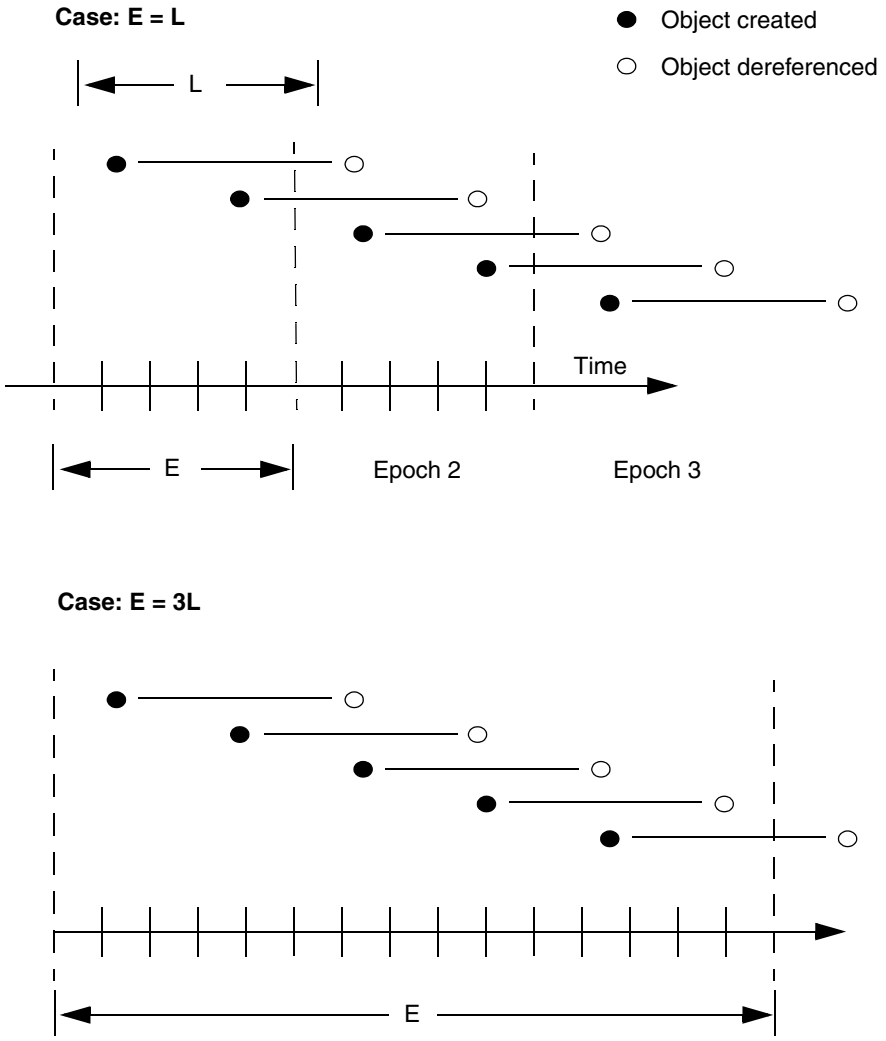
The results shown in Figure 11.5 can be expressed as:

Objects Missed by EpochGC = $R \times L$
Objects Recovered by EpochGC = $R(E - L)$

For example, assume R = 1000 objects per minute, L = 5 minutes, and E = 15 minutes. Then, for each epoch:

Objects Missed = 1000 x 5 = 5000
Objects Recovered = 1000 (15 – 5) = 10000

Figure 11.5 Effect of Collection Interval on Epoch Garbage Collection



Therefore:

- Set `#epochGcTimeLimit` *E* > lifetime *L* of short-lived objects.

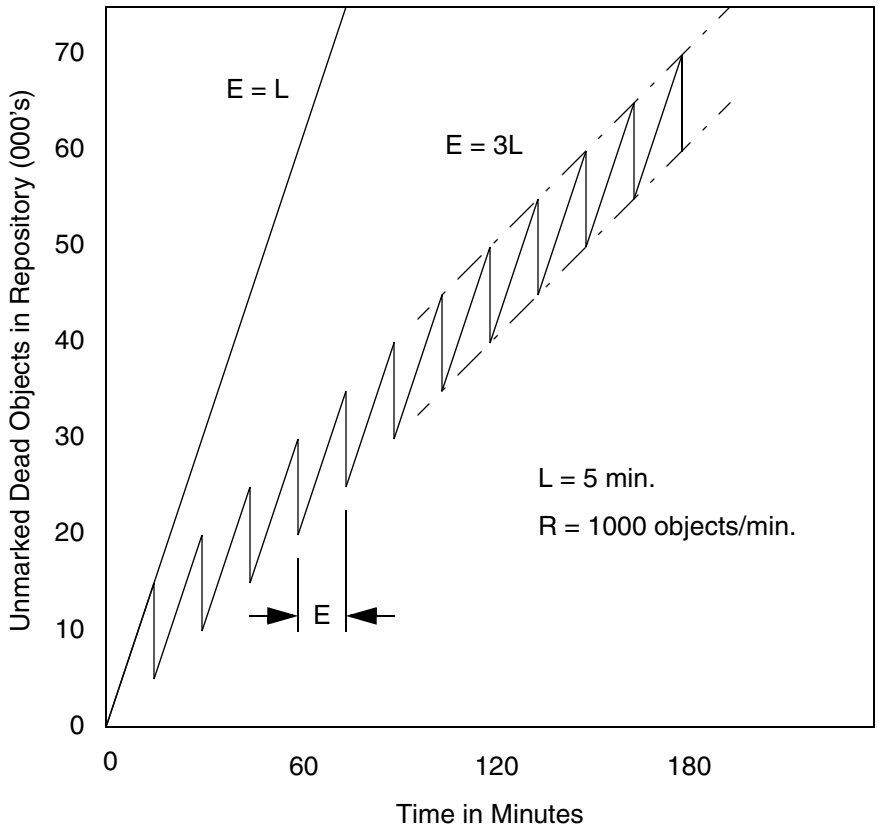
Figure 11.6 graphs the effect of the epoch. When $E = L$, epoch garbage collection is in effect disabled—all objects survive into the next epoch; the number of unmarked yet dead objects in the repository grows at the creation rate. These dead objects remain unidentified until you run `markForCollection`.

When the epoch is extended so that $E = 3L$, each epoch garbage collection marks those objects both created and dereferenced during that interval. This ratio causes the sawtooth pattern in the graph. If the creation rate is uniform, two-thirds of the dead objects are marked ($(E - L)/E$), and one-third are missed (L/E). Consequently, the repository grows at one-third the rate of the case $E = L$.

This configuration trades short bursts of epoch garbage collection activity for:

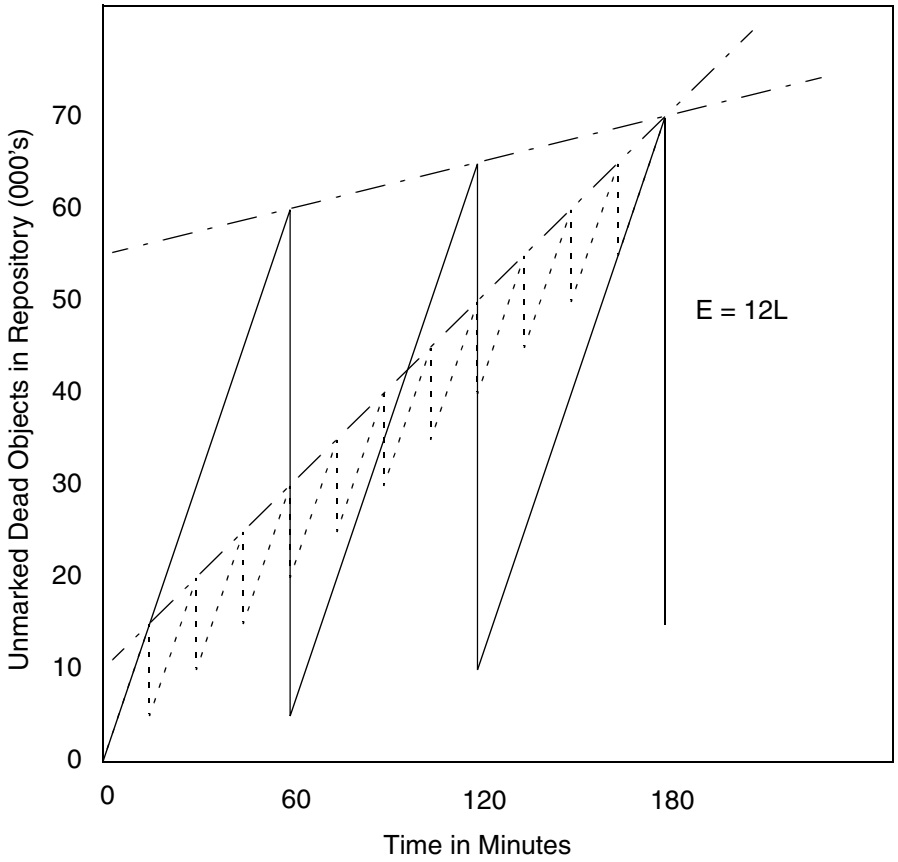
- moderate growth in the repository, and
- the need to run `markForCollection` often enough to mark dead objects that survive between epochs.

Figure 11.6 Repository Growth with Short Epoch



Suppose we extend the epoch to *E* = 12*L*. The result is shown in Figure 11.7, superimposed on part of the previous figure.

Figure 11.7 Effect of Longer Epoch on Repository Growth



Although the longer epoch allows many more dead objects to accumulate, the growth rate of the repository is substantially less—25% of the previous case.

This configuration trades a slower growth rate for:

- a need for greater headroom on the disk, and
- longer bursts of epoch garbage collection activity.

Certain cases have needed an epoch as long as several hours, or even a day.

Cache Statistics

Several cache statistics include information about the epoch garbage collection process. These are visible by using statmonitor data viewed in VSD (the visual statistics display tool); see Appendix G for details. You may also access methods in System to get the values programmatically; see "Programmatic Access to Cache Statistics" on page 174 for more information.

The following statistics may be useful in monitoring epoch:

EpochGcCount	The number of times that the epoch garbage collection process was run by the Admin GcGem since the Admin GcGem was started. For a system in steady state, look for uniform periods between runs or a uniform run rate.
EpochNewObjs	The number of new objects that were created during the last epoch.
EpochPossibleDeadObjs	The number of possible dead objects found by the last epoch garbage collection.
EpochScannedObjs	The number of objects scanned by the last epoch garbage collection.

11.5 Reclaim

The Reclaim GcGems are responsible for reclaiming both dead and shadowed objects (see "Shadow or Dead?" on page 306 for the difference between these types of garbage).

Shadowed objects are created naturally as your application modifies existing objects, so it is a good idea to have at least one Reclaim GcGem running to avoid shadowed objects accumulating. Some operations, such as migration, create a very large number of shadowed objects that need to be reclaimed.

After a mark/sweep operation — MFC, FDC/MGC, or epoch — completes, there will be a number of dead objects that need to be reclaimed.

While it is objects that are dead or shadowed, reclaim is done in pages. Pages that contain dead or shadowed objects may also contain some live objects; these live objects are copied to fresh pages, and the resulting page may then be reclaimed.

Reclaim occurs automatically in the background, provided the Reclaim GcGem is running, as configured by system configuration parameters and by the GcUser

configuration parameters. Although it is recommended to allow the background processes to perform the reclaim, you can explicitly invoke it by executing:

```
SystemRepository reclaimAll
```

Tuning Reclaim

Reclaim Configuration Parameters

The following configuration parameters are available to control the reclaim task, and are stored in GcUser's UserGlobals. For an example of how to change these parameters, see page 342.

`#reclaimSleepTime` The maximum amount of time in seconds that the process will sleep when no work is scheduled. Must be ≥ 1; the default is 10 seconds, maximum 3600.

`#sleepTimeBetweenReclaimMs`
The minimum amount of time in milliseconds that the process will sleep between reclaims, even when work is scheduled. The default is 0 milliseconds, maximum 3600000.

`#reclaimMinPages` The minimum number of pages to process in a single reclaim operation (reclaiming does not start until this threshold is reached). Must be ≥ 1; the default is 30 pages, maximum is SmallInteger maximumValue.

`#reclaimDeadEnabled` A Boolean indicating whether or not to reclaim dead objects; the default is true.

`#objsMovedPerCommitThreshold`
The approximate maximum number of live objects to move in a reclaim transaction. Must be ≥ 100; the default is 20000, maximum is SmallInteger maximumValue.

`#maxTransactionDuration`
The maximum length (in seconds) of a GcGem transaction. The transaction will be committed once this time is exceeded. Must be ≥ 10; the default is 300, maximum is SmallInteger maximumValue.

`#deadObjsReclaimedCommitThreshold`
The maximum number of dead objects to reclaim in a

single transaction, including dead objects reclaimed when reclaiming shadow pages. The default is 20000.

`#dataPageBufferSize`
: The buffer size, in pages, used to hold data pages to be read. Effectively controls the maximum number of pages read per single disk read. Changes to this parameter do not take effect until the Reclaim GcGem is restarted. Must be ≥ 10; the default is 16, maximum 128.

`#reclaimDeadShadowPageThreshold`
: The shadow page threshold at which dead objects are reclaimed. If there are fewer than this number of shadow pages to reclaim, then dead objects are also reclaimed. The default is 1000, maximum is SmallInteger maximumValue.

`#deferReclaimCacheDirtyThreshold`
: If the primary shared page cache (the shared cache on the stone's machine) is more than this percentage dirty, then reclaim gems will wait until the cache is less than 5% below this threshold before resuming reclaims. The default is 75%.

The reclaim Gem will commit as soon as any one of the following conditions is met:

- Number of live objects moved exceeds `#objsMovedPerCommitThreshold`.
- Duration of the transaction exceeds `#maxTransactionDuration`.
- Number of dead objects reclaimed exceeds `#deadObjsReclaimedCommitThreshold`.

Cache Statistics

Several cache statistics provide information about reclaim. These are visible by using statmonitor data viewed in VSD (the visual statistics display tool); see Appendix G for details. You may also access methods in System to get the values programmatically; see “Programmatic Access to Cache Statistics” on page 174 for more information.

The following Stone statistics may be useful in monitoring reclaim:

DeadNotReclaimedObjs
: The number of objects known to be dead but not yet reclaimed.

DeadObjsReclaimedCount
: The total number of dead objects reclaimed since the Stone repository monitor process was last started.

GcVoteState
: Indicates the current phase of garbage collection: Gems voting, voting complete, Possible Dead Write-Set Union Sweep (PDWSUS) in progress, or PDWSUS complete.

PagesNeedReclaimSize
: The amount of work waiting for the reclaim task.

PossibleDeadObjs
: The number of objects marked as dereferenced but not yet declared to be dead.

ReclaimCount
: The number of times the page scavenge (reclamation) process has been run.

ReclaimedPagesCount
: The number of scavenged pages.

11.6 Running the Admin GcGem

Privileges required: GarbageCollection

The Admin GcGem must be running to complete processing of dead objects following a `markForCollection` or `markGcCandidatesFromFile:` operation. If the Admin GcGem is not running following one of these operations, the garbage collection process cannot complete, and garbage can build up in your system.

Configuring the Admin GcGem

The Admin GcGem is configured to run by the default configuration file `system.conf`, and will be started automatically on Stone startup.

If you do not want to run the Admin GcGem, or if you have previously disabled the Admin GcGem and wish to reenable it, either edit the configuration file (by default, `system.conf`) before starting the Stone, or use runtime methods to set the configuration. If you set the configuration during runtime, your changes are

lost when the Stone repository monitor is subsequently restarted. For permanent changes, edit the configuration file.

Configure by Editing the Configuration File

The configuration parameter STN_ADMIN_GC_SESSION_ENABLED (page 375) specifies whether the Admin GcGem is started by the Stone at startup time.

Configure at Runtime

Before taking any action that requires the presence or absence of the Admin GcGem, you should verify whether the Admin GcGem is configured to run:

```
System configurationAt: #StnAdminGcSessionEnabled
```

If this method returns 1, the Admin GcGem is configured to run. If not, configure the Admin GcGem by executing:

```
System configurationAt: #StnAdminGcSessionEnabled put: 1.
```

Configuring the Admin GcGem to be enabled does not start it; starting the Admin GcGem requires a separate step.

Starting the Admin GcGem

To start the Admin GcGem (which you have already configured to run):

```
System startAdminGcSession
```

Alternatively, you can start the Admin GcGem together with all Reclaim GcGem sessions that are configured to run:

```
System startAllGcSessions
```

Both `startAdminGcSession` and `startAllGcSessions` initiate the start of the GcGems, and return immediately. The GcGems may take longer to actually initialize and start up.

To wait a given period of time for the Admin GcGem and all Reclaim GcGems to start up:

`System waitForAllGcGemsToStartForUpToSeconds:` *anInt*

This method starts all the GcGems that are configured, and waits for the specified number of seconds. If all the GcGems have not started up within that time, the method returns false. However, this does not necessarily mean that any GcGems have failed to start; on a slow system with a short timeout, this method may return false but all GcGems eventually start correctly.

To verify that the Admin GcGem is running:

```
System adminGcGemSessionId
```

This returns a SmallInteger representing the session ID of the Admin GcGem, or 0 if the Admin GcGem is not running.

To verify that the Admin GcGem is running, and that each extent has a Reclaim GcGem started:

```
System hasMissingGcGems
```

This returns false if the Admin GcGem is running and each extent has been assigned a Reclaim GcGem.

Stopping the Admin GcGem

To stop the Admin GcGem:

```
System stopAdminGcSession
```

To configure the Admin GcGem to not run:

```
System configurationAt: #StnAdminGcSessionEnabled put: 0.
```

This also stops the Admin GcGem, if it is running.

To stop the Admin GcGem and any Reclaim GcGem sessions that are running, use:

```
System stopAllGcSessions
```

11.7 Running Reclaim GcGems

When a Reclaim GcGem is running, it examines pages marked reclaimable because they contain either dead or shadow objects, within a single extent file or a specific range of extent files. It also reclaims fragments of space left by transactions that did not fill an entire page.

If the repository has multiple extents, multiple Reclaim GcGems may be running. Each Reclaim GcGem reclaims pages within its own extent or set of extents. You cannot have more than one Reclaim GcGem on any one extent, and you should not have any extents with no Reclaim GcGems associated.

Reclaimed space does not appear as free space in the repository until other sessions have committed or aborted all transactions concurrent with the reclaim transaction.

Two actions can hasten this moment:

- If your session is the only one logged in (other than GcGems), you can invoke page reclamation directly. See the following discussion, "Configuring Reclaim GcGems."
- If other users are logged in, you can determine which sessions are viewing the oldest commit record, thereby impeding reclamation. See the discussion, "Tuning Garbage Collection," on page 339.

Configuring Reclaim GcGems

One Reclaim GcGem is configured to run by the default configuration file `system.conf`, and will be started automatically on Stone startup.

To modify the number of Reclaim GcGems, you must configure the repository either by editing the configuration file (by default, `system.conf`) before starting the Stone or by using runtime methods to set the configuration. If you set the configuration during runtime, your changes are lost when the Stone repository monitor is subsequently restarted. For permanent changes, edit the configuration file.

Configure by editing the Configuration File

The configuration parameter STN_NUM_GC_RECLAIM_SESSIONS (page 386) specifies the number of Reclaim GcGems that are started by the Stone at startup time. You may not configure more Reclaim GcGems than the number of extents in the repository, as specified in DBF_EXTENT_NAMES (page 360). If you specify a value larger than the number of extents, one GC reclaim will be started for each extent.

Configure at Runtime

Before taking any action that requires the presence or absence of Reclaim GcGems, you should verify the number of Reclaim GcGems for which your repository is currently configured:

```
System configurationAt: #StnNumGcReclaimSessions
```

This method returns the number of Reclaim GcGems that are currently configured to run. These Reclaim GcGems may or may not be running.

When there are a high number of pages needing to be reclaimed, you might choose to shut down the single Reclaim GcGem during off-hours, and then start a larger number of Reclaim GcGems.

To update the number of Reclaim GcGems that are configured to run:

```
System configurationAt: #StnNumGcReclaimSessions put: N.
```

where *N* is the maximum number of reclaim GcGems that you intend to run. This method does not start the Reclaim GcGems.

To configure your system for the maximum number of Reclaim GcGems (one Reclaim GcGem per extent):

```
System configurationAt: #StnNumGcReclaimSessions put:
  SystemRepository numberOfExtents.
System startAllReclaimGcSessions.
```

Since reclaim places some load on heavily loaded systems (reclaimed pages must be committed), the performance impact of so many Reclaim GcGems may offset any potential advantage. Even when run during off-hours, a large number of Reclaim GcGems may cause commit bottlenecks for each other.

At least one Reclaim GcGem should be running at all times. In general, we recommend running one Reclaim GcGem for between 5 and 10 extents. You may need to experiment to find the correct balance for your system.

Configuring a Reclaim GcGem to be enabled does not start it; starting the Reclaim GcGem requires a separate step.

Starting Reclaim GcGems

To start all Reclaim GcGems that you have configured to run, execute one of the following methods:

```
System startAllReclaimGcSessions
```

This method starts the number of Reclaim GcGems that you have configured to run, dividing the extents evenly among them.

```
System startAllGcSessions
```

This method is similar to `startAllReclaimGcSessions`, but also starts the Admin GcGem.

Alternatively, you can explicitly assign Reclaim GcGems to a contiguous list of extents, as described in the next section, "Running Reclaim GcGems on a Range of Extents."

All of these methods initiate the start of the GcGems, and return immediately. However, the GcGems may take longer to actually initialize and start up.

To wait a given period of time for all the GcGems (Reclaim and Admin) to start up:

`System waitForAllGcGemsToStartForUpToSeconds:` *anInt*

This method starts all the GcGems that are configured, and waits for the specified number of seconds. If all the GcGems have not started up within that time, the method returns false. However, this does not necessarily mean that any GcGems have failed to start; on a slow system with a short timeout, this method may return false but all GcGems eventually start correctly.

Running Reclaim GcGems on a Range of Extents

A Reclaim GcGem can reclaim from a contiguous list of extents. You can control the allocation of Reclaim GcGems to extents using the method:

`System startReclaimGemForExtentRange:` *startExtent* `to:` *endExtent*.

For example, if there are four extents in the repository, and you have configured to run two Reclaim GcGems, to start one Reclaim GcGem on the first extent, and the other on the other three extents, the code would look like this:

```
topaz 1> printit
System startReclaimGemForExtentRange: 1 to: 1.
System startReclaimGemForExtentRange: 2 to: 4.
%
true
topaz 1> printit
System currentGcReclaimSessionsByExtent
%
an Array
  #1 3
  #2 6
  #3 6
  #4 6
```

As shown here, the method `currentGcReclaimSessionsByExtent` returns an array whose size is the number of extents in the repository (in this example, four). The element at each index is the session ID of the Reclaim GcGem for that extent.

Similarly, for a repository with 20 extents, you could set up four Reclaim GcGems as follows:

```
topaz 1> printit
System startReclaimGemForExtentRange: 1 to: 5.
System startReclaimGemForExtentRange: 6 to: 10.
System startReclaimGemForExtentRange: 11 to: 15.
System startReclaimGemForExtentRange: 16 to: 20.
%
true
```

Running Reclaim GcGems on Remote Nodes

You can configure Reclaim GcGems to run on remote nodes (nodes other than the one the Stone is running on). To do this, use the following method:

```
System class >> startReclaimGemForExtentRange: startExtent
to: endExtent onHost: hostNameOrIpAddress
```

For example, using the above example if the Reclaim GcGems should be run on a different node, the code might look as follows:

```
topaz 1> printit
System startReclaimGemForExtentRange: 1 to: 1 onHost:
'paris'.
System startReclaimGemForExtentRange: 2 to: 4 onHost:
'paris'.
%
true
topaz 1> printit
System currentGcReclaimSessionsByExtent
%
an Array
  #1 3
  #2 6
  #3 6
  #4 6
```

This code starts a remote shared page cache and page server on the remote machine, if they do not already exist. A GemStone NetLDI must be running on the remote node (for details, see page 94). Log files are written to the home directory of the user who owns the processes.

When the Stone is run on a multi-homed host (that is, a host with more than one IP address), you can use the following method to specify a `stoneHost:` argument:

`System class >> startReclaimGemForExtentRange:` *startExtent* `to:` *endExtent* `onHost:` *hostNameOrIpAddress* `stoneHost:` *stoneHostNameOrIpAddress*

This method allows the remote Reclaim GcGem sessions to communicate with the Stone on a separate network connection.

Verify the Reclaim GcGem Configuration

To verify if Reclaim GcGems are running, use one of the following methods:

```
System reclaimGcSessionCount
```

This returns the total number of Reclaim GcGems that are running.

```
System currentGcReclaimSessionsByExtent
```

This returns an array whose size is the number of extents in the repository. The element at each index in the array is the session ID of the Reclaim GcGem for that extent. (A zero indicates that there is no Reclaim GcGem for the corresponding extent.)

To ensure that all extents have Reclaim GcGems:

```
System numberOfExtentsWithoutGC
```

This returns the number of extents that will not have garbage collection performed, because there is no Reclaim GcGem associated with that extent.

```
System numberOfExtentRangesWithoutGC
```

This returns the minimum number of additional Reclaim GcGems that would need to be started in order to have all extents covered by a Reclaim GcGem.

To ensure that all extents have Reclaim GcGems, and that the Admin GcGem is running:

```
System hasMissingGcGems
```

Stopping the Reclaim GcGems

To stop all Reclaim GcGems that are currently running:

```
System stopAllReclaimGcSessions
```

To stop all Reclaim and Admin GcGems that are running:

```
System stopAllGcSessions
```

To configure Reclaim GcGems to not run:

```
System configurationAt: #StnNumGcReclaimSessions put: 0.
```

This also stops all Reclaim GcGems, if they are running.

11.8 Tuning Garbage Collection

Multi-Threaded Scan

For large systems, it can take a considerable amount of time to scan the entire repository, as is required by a mark/sweep operation (or other operations such as listInstances). To allow these scans to complete faster, operations that scan the entire repository use multiple threads running in parallel. There is a trade-off between how fast the operation completes and how much of the system resources it uses. Obviously, the faster the scan completes, the less of anything else can be done on that system during that period.

The trade-off can be configured per operation and can be changed as the operation proceeds. You can choose to run it to complete as quickly as possible but use all the system resources, or with minimal impact on the rest of your application, but taking much longer to complete. You can switch between these approaches as often as you need to.

Multi-threaded scan operations must be performed by Gems running locally to the Stone. When executed from a Gem that is running on a machine remote from the Stone, the scan will be performed by one thread, regardless of the number of threads requested. You can use **`System class>>sessionIsOnStoneHost`** to check that the gem is local.

Tuning Multi-Threaded Scan

Each session has two variables that control the impact of the multi-threaded operation it is running:

MtThreadsLimit	The upper limit on the number of threads can be activated.
MtPercentCpuActiveLimit	The total CPU load level at which the scan starts to deactivate threads.

These variables are arguments to all scan operations, although most scan operations have variants that use default values.

While these variables are passed in during operation startup, you can also update them while the scan is running. This enables you, for example, to reduce impact during working hours, while allowing more resources to be used during off hours.

Since the scan is running, of course, you need to update these variables from a second session, using the sessionId of the session that is running the scan.

One way to determine the session Id of the session that is running a scan operation is by checking the session holding the GcLock. However, while only one session can be holding the GcLock at a time, and markForCollection and FDC require the GcLock, other operations, such as GsObjectInventory, also may have the GcLock.

To access the upper limit on the number of threads:

```
System mtThreadsLimit: aSessionId
```

To update the upper limit on the number of threads:

```
System mtThreadsLimit: aSessionId setValue: anInt
```

To access the CPU load limit:

```
System mtPercentCpuActiveLimit: aSessionId
```

To update the CPU load limit:

```
System mtPercentCpuActiveLimit: aSessionId setValue: anInt
```

Both of these variables are used in tuning, but they have somewhat different uses. The primary way you will tune the impact on your system is by setting MtPercentCpuActiveLimit. The operations then controls its impact by activating or deactivating threads, up to a limit of MtThreadsLimit. The operation will proceed, using more or less resources at any particular time depending on what else is executing on your system. Note that the CPU load includes non-GemStone process running on this same machine, so if a machine is heavily used by non-GemStone processes, the operation may make little progress even if the GemStone repository itself is idle.

MtThreadsLimit acts as a ceiling on the impact as well. Since this limit is of more relevance within GemStone, on heavily loaded machines you may want to pay more attention to this limit to control the impact within the repository. This limit is also useful when you want to pause the scan. Setting the MtThreadsLimit to 0 means that the scan cannot perform work, but does not stop executing, it waits until a non-zero limit s set.

Cache Statistics

The following cache statistics are important for tuning multi-threaded scans. These are visible by using statmonitor data viewed in VSD (the visual statistics display tool); see Appendix G for details. You may also access methods in System to get the values programmatically; see "Programmatic Access to Cache Statistics" on page 174 for more information.

MtThreadsLimit — The upper limit on the number of threads that can be running at any one time.

MtPercentCpuActiveLimit — The upper limit on percent of CPU that can be active before threads are deactivated.

percentCpuActive — The current percentage of CPU that is active.

MtActiveThreads — The current number of active threads

Memory Impact

Multi-threaded operations may require considerable heap memory. This memory requirement is **not** part of temporary object cache memory. You can configure your GEM_TEMPOBJ_CACHE_SIZE according to other application Gem requirements, or even configure the sessions performing repository scan operations with a very small temporary object cache size.

The amount of memory space that is needed depends primarily upon the current oopHighWater value, the number of threads, and the page buffer size. markForCollection uses a pageBufferSize of 128, epoch and writeSetUnionSweep use a size of 64, and it is an explicit argument to FDC.

The overhead associated with the oopHighWater value can be computed as:

```
(stnOopHighWater + 10M) / 2
```

The memory cost per thread is:

```
50K + (180K * pageBufSize)
```

For example, a system with an oopHighWater mark of 500M running eight threads with a page buffer size of 128 would require a minimum of about 440 MB of free memory.

GcGem Configuration Parameters

Tuning garbage collection operations is done by adjusting configuration parameter settings that are stored in GcUser's UserProfile. To modify them, log in

as GcUser (GcUser's default password is the same as DataCurator's or SystemUser's default password) and send the message `at:`*aKey* `put:`*aValue* to UserGlobals. For example, to set `#reclaimMinPages` to 100:

```
topaz> set user GcUser password thePassword
login
...
topaz 1> printit
UserGlobals at: #reclaimMinPages put: 100.
System commitTransaction
%
```

There may be a delay in the GcGems seeing changes to GcUser's configuration parameters if the GcGem is in a long-running operation, such as the write set union sweep.

While most GcUser parameters are specific to operations performed by either the Admin GcGem or Reclaim GcGems, the following are used by both:

`#verboseLogging` Determines if long explicit messages are printed to the GcGem log files. The default is false.

`#autoRefreshGcGemConfig` Determines whether the GcGems re-read their configuration settings from UserGlobals at the end of each transaction. The default is true.

`#enableDebugging` Activates extra consistency checks in the GcGems used for debugging. The default is false.

Additional configuration parameters for the GcGem are discussed on page 321 for tuning epoch garbage collection, page 329 for tuning reclaim, and page 342 for tuning the write set union sweep.

Tuning Write Set Union Sweep

The write set union sweep is phase 6 of garbage collection, as described on page page 310. It is performed by the Admin GcGem.

The write set union sweep is performed using the multi-threaded scan (page 339), and can be tuned using the following GcGem parameters:

`#SweepWsUnionPercentCpuActiveLimit` Limit active wsUnion threads when system percentCpuActive is above this limit. Default: 90.

`#SweepWsUnionPageBufferSize`
: Size (in pages) of buffer used for wsUnion sweep. Must be a power of 2. Default: 64. Minimum: 8. Maximum: 1024.

`#SweepWsUnionMaxThreads`
: The maximum threads used for next wsUnion sweep. By default, use one thread.

Identifying Sessions Holding Up Page Reclamation

Privileges required: SessionAccess.

Reclaiming pages can proceed only up to those pages currently providing some session's transaction view of the repository—that is, only up to the oldest commit record. When other sessions are logged in, reclamation stops at that point until all sessions using that commit record either commit or abort their transaction.

It can be helpful to identify which sessions are holding on to the oldest commit record. The method `System class>>sessionsReferencingOldestCr` returns an array of session IDs, which can be mapped to GemStone logins through `System class>>currentSessionNames` or `System class>>descriptionOfSession:`*aSessionId*. For example:

```
topaz 1> printit
System sessionsReferencingOldestCr
%
an Array
  #1 5

topaz 1> printit
System currentSessionNames
%
session number: 2    UserId: GcUser
session number: 3    UserId: GcUser
session number: 4    UserId: SymbolUser
session number: 5    UserId: DataCurator
session number: 6    UserId: DataCurator
```

The method `descriptionOfSession:` is particularly useful in that it returns an array of descriptive information. The second element is the operating system process ID (pid), and the third element is the name of the node on which the process is running. For details, see the comment in the image.

Removing References to Large Objects

If you know that you have large objects that are no longer needed, another way to free space is to explicitly remove references to them. To remove such objects, you must first identify them. Then you can find all references to them and remove those references.

Identify Large Objects in the Repository

The following expression causes GemStone to look through the symbol list for each user in AllUsers and gather information on any named objects larger than the SmallInteger *aSize*.

```
topaz 1> printit
AllUsers findObjectsLargerThan: aSize limit: aSmallInt
%
```

This method locates large collections or strings stored directly in the Symbol Lists, for example, as created by an expression such as

```
UserGlobals at: #aCollection put: IdentityBag new
```

It will not locate collections stored within the class variables of classes or stored in instances of classes.

It returns an Array of up to *aSmallInt* elements, each of the form:

```
{ { aUserId . aKey . anObject } }
```

where *anObject* is an object larger than *aSize* defined in the symbol list of *aUserId*, and *aKey* is the Symbol associated with that object.

If any references to anObject are protected by a GsObjectSecurityPolicy for which you do not have read authorization, the last element of the result is the String 'Read Authorization Error encountered'.

With the information from this method, you may be able to track down temporary collections that were inadvertently committed, by checking the SymbolLists of the given user and locating an object referenced by the given key. However, if there are large objects that are not stored directly in symbol lists, you can send the same message to System to perform a global search.

The following method which will return any large objects, regardless of where they are stored:

```
topaz 1> printit
System findObjectsLargerThan: aSize limit: aSmallInt
%
```

This returns an Array of all objects in the repository larger than the SmallInteger *aSize*, whether they are named in a user's symbol list or not. As above, the Array is limited to a maximum of *aSmallInt* elements.

Again, if any references to anObject reside in ObjectSecurityPolicies for which you do not have read authorization, the last element of the result is the String 'Read Authorization Error encountered'.

This method performs the repository wide search using multi-threaded scan; see page 339 for a description. **`findObjectsLargerThan:limit:`** uses one thread and a CPU limit of 100%, so it may have some impact, particularly on smaller host systems, and may take some time to complete.

To complete the operation as quickly as possible, using as much of the system resources as necessary, you can use the similar method `System class >> fastFindObjectsLargerThan:limit:`.

Search for References to an Object

Full reference path

To find a complete reference path to a particular object, you may use the method `Repository >> findReferencePathToObject:`. This method reports the complete reference path from a root object to the argument.

NOTE

This method runs in transaction, may take a considerable time to run. Avoid using it in production systems.

The result of this method is an array, with the first two elements consisting of the search object and true or false indicating if a reference path was found. No reference path means that the object would be garbage collected by the next MFC or FDC/MGC cycle. If a reference path was found, following this in the array are the list of objects that connect a root object to the search object, with the search object again appearing at the end, the reference to the search object in the second to last position, and so on.

For example,

```
topaz 1> run
SystemRepository findReferencePathToObject: AllDeletedUsers
%
<RefPathScan information>
an Array
  #1 an IdentitySet
  #2 true
  #3 a SymbolDictionary
  #4 an IdentityCollisionBucket
  #5 a SymbolAssociation
  #6 an IdentitySet
```

AllDeletedUsers is an IdentitySet (#1 and #6). It is referred to in a SymbolDictionary (#3), using internal implementation objects (an IdentityCollisionBucket and a SymbolAssociation) that actually have the references.

This method returns the first path from a root object to the argument object that is found, but there may be multiple paths. You can perform reference path searches for multiple objects at the same time, or for multiple paths for the objects, using the method:

```
topaz 1> run
SystemRepository findReferencePathToObjects: arrayOfObjects
   findAllRefs: aBoolean printToLog: aBoolean
%
```

This returns an array of data, one element for each object in *arrayOfObjects*, containing the reference path information.

All References

You can search the repository for multiple references to an object by sending the following message:

```
topaz 1> printit
anObject findReferencesWithLimit: aSmallInt
%
```

This returns an Array of objects in the repository that reference *anObject*. If an object contains multiple references to *anObject*, that object will appear only once in the resulting Array. The Array is limited to a maximum of *aSmallInt* elements, to limit the duration and memory requirements of the result.

The resulting Array contains only those references that are protected by GsObjectSecurityPolicies for which you have read authorization. If any references to *anObject* are protected by GsObjectSecurityPolicies for which you do not have read authorization, the last element of the result is the String 'Read Authorization Error encountered'.

`findReferencesWithLimit:` uses multi-threaded scan (see page 339 for details). It uses a default of two threads and up to 90% of the CPU.

Remove References to an Object

Complete the process by replacing references to the unneeded object with `nil`. This allows the object to be removed during normal garbage collection. Dereferencing objects must be done by Smalltalk code execution.

Appendix

A GemStone Configuration Options

A GemStone/S 64 Bit configuration file is a file containing information that, when read at startup time, can control the configuration, behavior, and functionality of the system at run time. Some of these configuration settings can be modified dynamically by using GemStone Smalltalk to change their internal representation.

The Stone, Gem, and linked applications (collectively, the repository executables) are able to read two different types of configuration files: system-wide configuration files and executable-dependent configuration files.

- **System-wide configuration files** allow the GemStone system administrator to set options pertaining to all GemStone executables on a system- or network-wide basis. This file is required for a Stone to start.

  System-wide configuration files are found in a default location, passed by argument, or located by a GEMSTONE_SYS_CONF environment variable.

- **Executable-dependent configuration files** can be used by individual users to control their own running copy of the GemStone system. Options contained in executable-dependent configuration files override the options specified in a system-wide configuration file.

  Executable-dependent configuration files are found by name, passed by argument, or located by a GEMSTONE_EXE_CONF environment variable.

These environment variables can be set in the usual way. For example:

```
$ GEMSTONE_EXE_CONF=$HOME/myFile.conf
$ export GEMSTONE_EXE_CONF
```

Both GEMSTONE_SYS_CONF and GEMSTONE_EXE_CONF can be defined to point to either a file or a directory.

The GemStone executables **startstone, pageaudit, topaz,** and **gemsetup** accept command line arguments to point to configuration files:

- The **-z** option sets the system configuration file.
- The **-e** option sets the executable-dependent configuration file.

A.1 How GemStone Uses Configuration Files

At start-up time, GemStone repository executables attempt to find and read both a system-wide and an executable-dependent configuration file, searching for these files in the following manner.

Search for a System-Wide Configuration File

GemStone repository executables begin by attempting to find a system-wide configuration file.

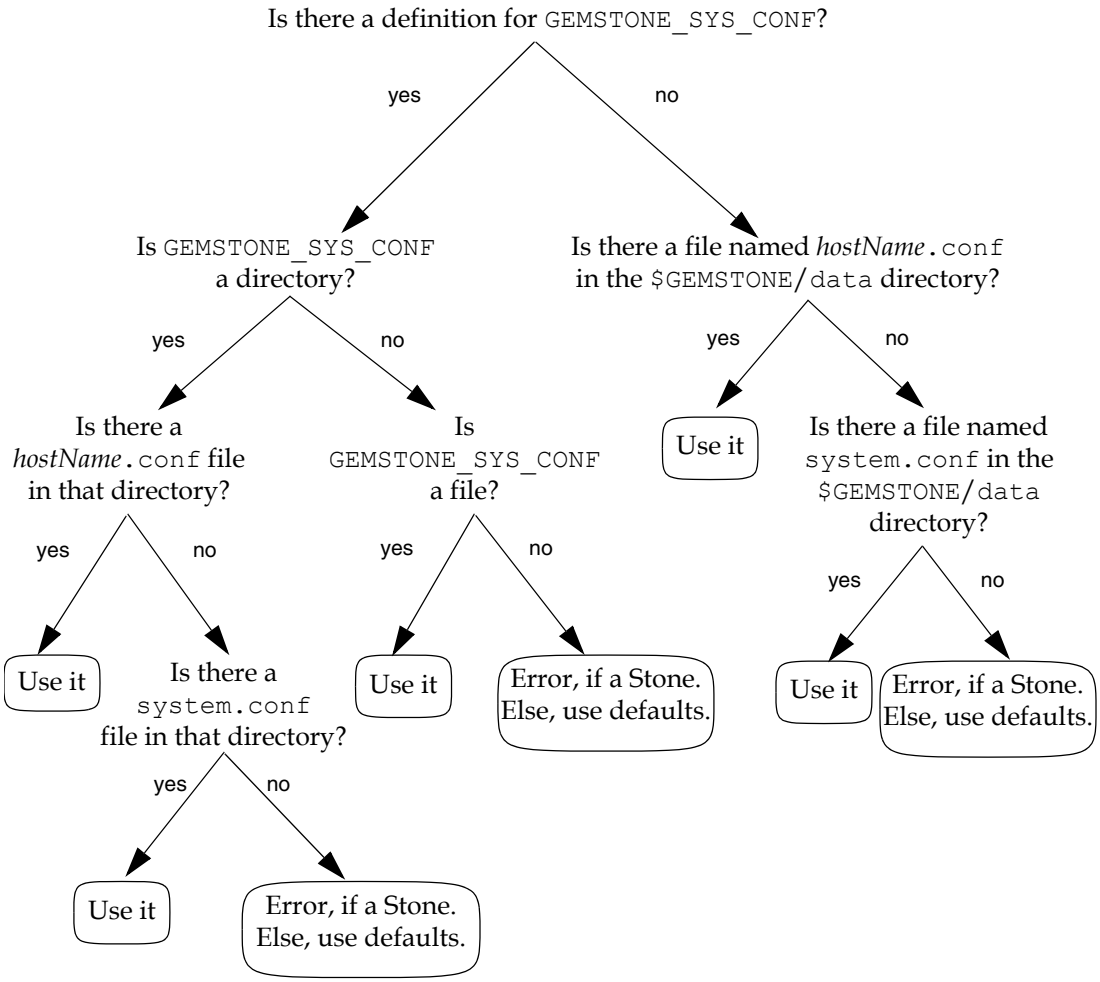
1. As shown in Figure A.1, GemStone first checks to see if there is an environment variable defined for GEMSTONE_SYS_CONF.
2. If GEMSTONE_SYS_CONF is *not* defined, GemStone looks for a file named *hostName*.conf in $GEMSTONE/data and uses that file. *hostName* must match the results of executing the hostname command on the machine on which the executables are running.
3. If no such file exists, it looks for a file named system.conf in $GEMSTONE/data and uses that.
4. If neither of those files exist, the system defaults are used, unless the executable is Stone, in which case an error is generated.
5. If GEMSTONE_SYS_CONF *is* defined, GemStone checks to see if it points to a directory.
6. If GEMSTONE_SYS_CONF points to a directory, GemStone looks for a file named *hostName*.conf in that directory. If it finds such a file, it uses it. *hostName* must match the results of executing the hostname command on the

machine on which the executables are running. If no *hostName*.`conf` is found, it looks in that directory for a file named `system.conf` and uses that. If neither of those files exist, the system defaults are used, unless the executable is Stone, in which case an error is generated.

- If the GEMSTONE_SYS_CONF environment variable points to a file instead of a directory, GemStone just uses that file.

Within each file, if an option is listed more than once, then the value it is given the last time it is specified is used as its true value at executable run time. This rule also applies between the two types of configuration files. If the same option is given a value in both the system-wide and executable-dependent configuration files, the value in the executable-dependent configuration file overrides the system-wide configuration file's value.

Figure A.1 Search Path for a System-Wide Configuration File

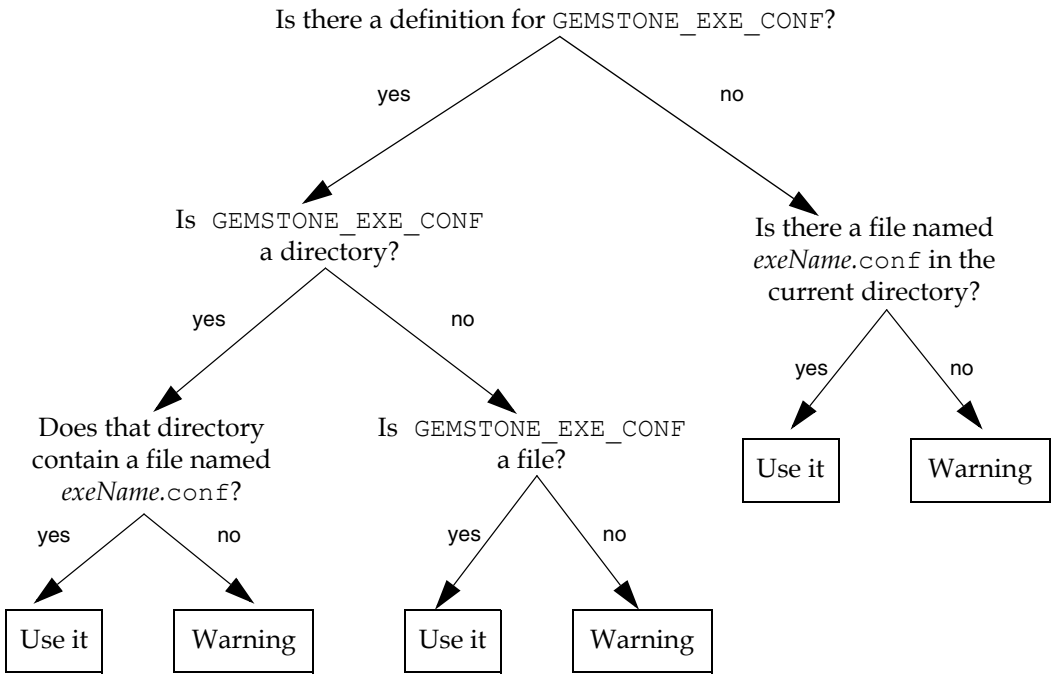


Search for an Executable Configuration File

Ordinarily, GemStone repository executables next try to find an executable-dependent configuration file. (The exception is a Stone repository monitor that failed to find its system-wide configuration file—it exits with an error.)

1. As shown in Figure A.2, GemStone begins by checking to see if there is an environment variable defined for GEMSTONE_EXE_CONF.
2. If GEMSTONE_EXE_CONF is not defined, GemStone tries to find a file called *exeName*.conf in the current working directory. (For information about the naming conventions, see "Naming Executable Configuration Files" on page 355.)
3. If it succeeds at finding such a file, it uses that file. If such a file does not exist, it generates a warning and relies solely on the system-wide configuration file for configuration parameters.

Figure A.2 Search Path for an Executable-Dependent Configuration File



If GEMSTONE_EXE_CONF *is* defined, GemStone first looks to see if it points to a directory.

- If GEMSTONE_EXE_CONF points to a directory, GemStone looks for a file named *exeName*.conf in that directory. If such a file exists, it uses it; if not, a warning is generated and GemStone relies on the system-wide configuration file for configuration parameters.
- If GEMSTONE_EXE_CONF points to a file, rather than to a directory, GemStone simply uses that file.
- If GEMSTONE_EXE_CONF points to a directory or file that doesn't exist, a warning is generated and GemStone defaults to using the system-wide configuration file for configuration parameters.

Creating or Using a System Configuration File

If you are satisfied with the standard options and the defaults, the simplest thing to do is to just use the configuration file provided in `$GEMSTONE/data/system.conf`. You can either copy this file and set the GEMSTONE_SYS_CONF environment variable to point to your new file, or you can do nothing and let GemStone use `$GEMSTONE/data/system.conf` itself.

Creating an Executable Configuration File

There are two ways to create a configuration file for a specific executable:

- You can copy the entire system-wide configuration file to a new file, name it appropriately, and change selected parameters.
- You can create a new file, give it an appropriate name, and include only those parameters that you want to differ from the default.

To make sure that GemStone is able to find and use your executable-dependent configuration file, you can set the GEMSTONE_EXE_CONF environment variable to point to your file. GEMSTONE_EXE_CONF can be either a file name or a directory name. If you set the environment variable to a directory name, be sure to name the configuration file *exeName*`.conf` so GemStone can find it at start up. (Information about the naming conventions for configuration files is just ahead.)

If you don't set the GEMSTONE_EXE_CONF environment variable, GemStone looks for a file named *exeName*`.conf` in the current working directory at startup. If

it doesn't find one, it uses the configuration parameters set in the system-wide configuration file, or it uses the system defaults.

NOTE

Make sure your executable-dependent file is both readable and writable by the Stone process, which will update options by writing to it if you make certain configuration changes at run time.

Naming Executable Configuration Files

The default name of an executable configuration file generally is determined from the name of the executable itself.

Application Gems

Stand-alone (RPC) Gems look for a file named `gem.conf` in the current working directory unless GEMSTONE_EXE_CONF is defined. The working directory by default is the user's home directory, unless the `gemnetobject` script has been customized. The file `$GEMSTONE/sys/gemnetobject` is a script that a NetLDI invokes to start a GemStone session process. This script can be edited to define the name of the Gem to execute, the directory where the Gem resides, and the GEMSTONE_SYS_CONF and GEMSTONE_EXE_CONF environment variables.

System Gems

It is sometimes useful to change the parameters in a configuration file specific to a system Gem, such as the Admin or Reclaim GcGems, the Page Manager, Symbol Gem, or cache warmer Gems. This allows customized configuration settings that remain in effect if the system is stopped and restarted.

To do so:

Step 1. Copy $GEMSTONE/`data/system.conf`.

Step 2. Edit the copy, setting the values you want.

Step 3. Save your changes, renaming the file appropriately (for example, `admingcgem.conf`, `reclaimgcgem.conf`, etc., depending on which system Gem the new configuration file is for). Place the file in GemStone's `sys` directory.

Step 4. Make a copy of the appropriate script/s.

`runadmingcgem`	Starts the Admin GcGem.
`runcachewarmergem`	Starts the cache warmer Gems.
`runotcachewarmergem`	Starts the Object Table cache warmer Gems.
`runpagemgrgem`	Starts the page manager Gem.
`runreclaimgcgem`	Starts the Reclaim GcGem.
`runsymbolgem`	Starts the SymbolGem.

These scripts are located in $GEMSTONE/`sys/`. Name the copy appropriately; for example, $GEMSTONE/`sys/my`**`runreclaimgcgem`**.

Step 5. Edit `myrunreclaimgcgem` to specify the customized configuration file. For example, to use a Reclaim GcGem configuration file named `$GEMSTONE/sys/reclaimgcgem.conf`, locate the lines:

```
exeConfig=""
```

and change to:

```
exeConfig="$GEMSTONE/sys/reclaimgcgem.conf"
```

Step 6. Edit the file `$GEMSTONE/sys/services.dat`. Comment out the existing line:

```
runreclaimgcgem      $GEMSTONE/sys/runreclaimgcgem
```

and add an entry specifying the new script to be executed for the `runreclaimgcgem` service.

```
#runreclaimgcgem     $GEMSTONE/sys/runreclaimgcgem
runreclaimgcgem      $GEMSTONE/sys/myrunreclaimgcgem
```

Stone

Stone looks for a file named *stoneName*`.conf` in the current working directory.

Linked Topaz

The linked version of Topaz looks for the configuration file `gem.conf` so, by default, Gem and Topaz can share the same options. The default location of the file is the user's home directory (`$HOME`).

Linked GBS

Linked GBS logins by default look for a file named `gbs.conf`. The default location of the file is the user's home directory (`$HOME`).

Linkable GemBuilder for C Applications

Linkable GemBuilder for C applications look for a file named `gci.conf` in the current working directory unless the application has provided a different name by calling GciInitAppName().

Naming Conventions for Configuration Options

The prefix "GEM_" indicates that the option is processed directly by Gems. Unless indicated otherwise by the phrase "used by all executables," most other options are processed only by the Stone, which passes the information to executables as needed through network connections. Exceptions are the shared page cache configuration options ("SHR_"). The first Gem session process on a node remote from the Stone and extents reads these options, which determine the configuration of the shared page cache on that node.

All executables (that is, the Stone and Gems) understand the standard options used in the file `$GEMSTONE/data/system.conf` as shipped. The GemStone executables generate a warning message whenever they encounter an option that is not in the standard list.

NOTE:
If the DUMP_OPTIONS *option is set to true, then once the system-wide and executable-dependent configuration files have been processed, the values of all the options understood by the executable are displayed. You can access the configuration parameters from Smalltalk by using the methods described starting on page 62.*

A.2 Configuration File Syntax

The following section describes the rules of grammar to be used in editing configuration files.

White space Leading white space is ignored in the parsing of configuration files. Trailing white space is ignored if it follows the statement termination symbol (;).

New lines New lines within a statement are allowed only after an equal sign or after a comma within a list of values.

Comments The comment symbol for GemStone configuration files is the pound sign (#).
Comments can be embedded in a configuration file using the following rules:

- by starting a line with the comment symbol
- by placing any text after the statement termination symbol (;)

Lists

Lists are separated by commas; list elements can be empty, for example:

```
DBF_EXTENT_SIZES = 1999, , 1999;
```

Within lists of values, leading and trailing white space is ignored.

Strings

Strings are encased in quotes. An empty string is acceptable in the grammar, and may be expressed by either two double quotes ("") or by no value at all (for instance, OPTION = ;).

Within strings, the escape character is the backslash (\). It can be used as follows:

To generate:	Use the sequence:
backslash (\)	\\
quote (")	\"
statement termination symbol (;)	\;
list separation character (,)	\,
control characters	\ followed by decimal representation of the character as a zero-padded 3-digit decimal number. For example, the string control-N would read \014, because control-N is ASCII 14.

Case-sensitivity

String option values are case-sensitive; boolean option names are *not* case-sensitive.

Maximum Sizes

The maximum number of characters allowed for a GemStone configuration option name is 64. The maximum length of a string option is 1024 characters. There is no limit on the number of elements within a list.

Use of Environment Variables In Options

Options that are either file names or directories may have environment variables as the first part of their value or the

entire value. For instance,
`$GEMSTONE/data/extent0.dbf`.

Errors in Configuration Files

At startup, each GemStone executable reads the configuration files. If any error is detected, information about the error is written to the standard output. This information indicates the file and line containing the error and the error's severity.

Two kinds of errors can be generated by the processing of configuration files: syntax errors and option value errors.

Syntax Errors

Syntax errors are generated whenever a grammatical error is detected in the configuration file. All syntax errors are warnings; they do not cause execution to terminate. These errors include:

- End-of-line or end-of-file detected before expected
- Invalid starting character for an option name or invalid character within an option name
- Equals or semicolon sign expected
- Invalid 3-digit escape sequence
- Invalid escape character
- Terminating quote missing in a quoted string

Option Value Errors

Option value errors are generated when the value assigned to an option has no meaning or is of the wrong type. For example, an option value error is generated when an option defined to need a boolean for its value has been set to an integer.

Option value errors vary in severity. Some options, such as not specifying the list of files that make up a logical repository, will necessarily terminate execution. Other option value errors, such as a invalid cache size, might only generate warnings. When a warning is issued, the executable ignores the given value and use the option's default value.

A.3 Configuration Options

The system-wide configuration file contains the following standard configuration options. In this discussion, *default* refers to the value that results when an option is not explicitly set by a statement in the configuration file. *Initial setting* refers to an explicit setting in the initial `system.conf` file that differs from the default.

Some configuration options have an internal runtime parameter that can be changed while GemStone is running. Where such a parameter exists, its name is given as part of the entry. For more information, see "To Change Settings at Run Time" on page 63.

Note that the `$GEMSTONE/bin` directory contains a write-protected file named `initial.config` that is an exact replicate of `$GEMSTONE/data/system.conf` as it was originally shipped, so even if you change the `system.conf` file, you can always recover its original condition.

The following configuration options are listed in alphabetical order.

DBF_ALLOCATION_MODE

DBF_ALLOCATION_MODE describes the space allocation heuristic to be used when filling repository extents.

Permissible values are either Sequential or a series of allocation weights, separated by commas. Under sequential allocation, each extent has its full resources used before the next extent's resources are used. Under weighted allocation, those extents with a larger weight will have proportionally more of their disk resources allocated than those with smaller weights. Each weight applies to the corresponding extent in the series of extents specified in DBF_EXTENT_NAMES, and the number of elements must match. Extent allocation weights must be integers in the range 1..40 (inclusive).

Default: Sequential

DBF_EXTENT_NAMES

DBF_EXTENT_NAMES list of all repository extents, in order, primary extent first, separated by commas. Taken together, all of the listed file resources make up the logical repository. This option is required, and must contain at least one entry, the name of the primary extent. The maximum number of extents is 255.

An extent name can be a file name or the device name for a raw disk partition. The name can have an environment variable as its first component.

Default: EMPTY. The system will not run unless you define an extent list.
Initial setting: `$GEMSTONE/data/extent0.dbf`

DBF_EXTENT_SIZES

DBF_EXTENT_SIZES sets the maximum sizes (in MB) of all repository extents, in order, primary extent first, separated by commas. Each size applies to the corresponding extent in the series of extents specified in DBF_EXTENT_NAMES.

A size entry may be null, which indicates that the corresponding extent has no fixed maximum size. This setting allows the extent to grow until it fills the disk containing it or until it reaches the maximum size for an extent.

NOTE

The maximum size of an extent is operating system- and platform-dependent, but under no circumstances can be larger than 32 terabytes (32,000,000 MB). For specific information about system dependencies, see the comment in the configuration file for the parameter `DBF_EXTENT_SIZES`.

For optimal performance using a raw partition, DBF_EXTENT_SIZES should be slightly smaller than the size of the partition so GemStone can avoid having to handle system errors. For example, set it to about 1995 MB for a 2 GB partition.

You can modify the size of an existing extent under these conditions:

- If the original maximum size was unlimited, the new maximum size must be larger than the current physical size of the extent.
- If the original maximum size was limited, the new maximum size must be larger than the original maximum size.

The Stone repository monitor is the only executable allowed to change DBF_EXTENT_SIZES. At GemStone system startup, the maximum size of each extent is written to the system log file.

Default: EMPTY (no maximum sizes)
Min: 1 (MB)
Max: operating system- and platform-dependent (see NOTE above)

DBF_PRE_GROW

If DBF_PRE_GROW is set to true, then when a new extent is created, it is grown to its maximum size. If the new extent cannot be grown to the maximum size because of disk capacity problems, then the creation will fail.

The default value for DBF_PRE_GROW is false. This setting indicates that extents will grow only when new space is needed by the logical repository.

An extent without a maximum size is not pregrown to any size; it is allocated a minimum size determined internally by the GemStone system.

DBF_PRE_GROW, if set to true, will affect existing extents at startup. If an existing extent has a maximum size and that extent is physically not at that maximum size, it is grown to that size and the added portion is initialized. If the grow fails, the extent is reset to its original size and the startup attempt fails.

Default: false

DBF_SCRATCH_DIR

DBF_SCRATCH_DIR specifies a scratch directory that the Stone process can use to create "scratch" repositories for use during **pageaudit**. The file name is appended to the directory name *without* an intervening delimiter, so a trailing delimiter is necessary here.

Default: `$GEMSTONE/data/`

DUMP_OPTIONS

If DUMP_OPTIONS is set to true, dumps a summary of all configuration options.

Default: true

GEM_ABORT_MAX_CRS

When a Gem is not in a transaction and aborts, GEM_ABORT_MAX_CRS specifies the maximum number of commit records to analyze to compute the writeSetUnion since the last time this session aborted. If the number of commit records would exceed this limit, the abort is treated similar to a LostOt and all in-memory copies of committed objects are marked invalid and will be re-read as needed during subsequent execution.

A value of 0 (zero) means no limit on number of commit records to analyze.

Min: 0
Max: 2147483647
Default: 0

GEM_FREE_FRAME_CACHE_SIZE

GEM_FREE_FRAME_CACHE_SIZE specifies the size of the Gem's free frame cache. When using the free frame cache, the Gem removes enough frames from the free frame list to refill the cache in a single operation. When adding frames to the free list, the Gem does not add them until the cache is full.

A value of 0 disables the free frame cache (the Gem acquires frames one at a time). A value of –1 means use the default value: 0 for caches less than 100 MB and 10 for caches of 100 MB or greater.

Cache Statistic: FreeFrameCacheSize (Gem)
Units: frames
Min: –1
Max: 63
Default: –1 (cache size=0 for caches less than 100 MB and 10 for caches of 100 MB or greater)

GEM_FREE_FRAME_LIMIT

When the number of free frames in the shared page cache is less than GEM_FREE_FRAME_LIMIT, the Gem session process scans the cache for a free frame rather than using one from the free frame list. This action is desirable for performance reasons so the remaining frames in the list are available for use by the Stone repository monitor.

If the value of GEM_FREE_FRAME_LIMIT is –1, the free frame limit is set to one of the following default values:

- For primary shared page cache that is 800 MB or smaller: 10% of the number of frames in the cache
- For primary shared page cache greater than 800 MB: 5000 frames
- For a remote shared page cache: 0

Runtime parameter: **#GemFreeFrameLimit**
Cache Statistic: FreeFrameLimit (Gem)
Min: 1
Max: 65536
Default: –1 (see above discussion)

GEM_FREE_PAGEIDS_CACHE

GEM_FREE_PAGEIDS_CACHE specifies the maximum number of free pageIds to be cached in Gem. Larger values reduce number of calls to Stone, at a cost of needing more free space within the extents.

Runtime parameter: **#GemFreePageIdsCache**
Min: 40
Max: 3500
Default: 200

GEM_GCI_LOG_ENABLED

This option has no effect in customer executables.

Default: false

GEM_HALT_ON_ERROR

GEM_HALT_ON_ERROR causes a Gem to halt and dump core if an error with the specified GemStone error number occurs. The value 0 means "never halt". Ordinarily this option is used only to assist Technical Support in diagnosing problems.

Default: 0

GEM_KEEP_MIN_SOFTREFS

GEM_KEEP_MIN_SOFTREFS determines the minimum number of most recently used SoftReferences that will not be cleared by VM markSweep if *startingMemUsed* — the percentage of temporary object memory in use at the beginning of a VM mark/sweep — is greater than GEM_SOFTREF_CLEANUP_PERCENT_MEM but less than 80%.

In most cases, the default (0) is appropriate and should not be changed.

Runtime parameter: **#GemKeepMinSoftRefs**
Min: 0
Max: 10000000
Default: 0

GEM_MAX_SMALLTALK_STACK_DEPTH

GEM_MAX_SMALLTALK_STACK_DEPTH determines the size of the GemStone Smalltalk execution stack space that is allocated when the Gem logs in. The unit is the approximate number of method activations in the stack. This setting causes heap memory allocation of approximately 64 bytes per activation. Exceeding the stack depth results in generation of the error RT_ERR_STACK_LIMIT.

Min: 100
Max: 1000000
Default: 1000

GEM_NATIVE_CODE_ENABLED

If GEM_NATIVE_CODE_ENABLED is true, enables the generation and use of native code.

The runtime parameter #GemNativeCodeEnabled can be used to disable native code, but cannot be used to re-enable native code once it has been disabled during a session.

Runtime equivalent: **#GemNativeCodeEnabled**
Default: TRUE.

GEM_PGSVR_COMPRESS_PAGE_TRANSFERS

If GEM_PGSVR_COMPRESS_PAGE_TRANSFERS is true, use compress2() from zlib library with default compression level to compress page transfers between the page server on Stone's machine and Gem or mid-cache page server.

For the first Gem to login on a remote machine, that Gem's configuration file value of GEM_PGSVR_COMPRESS_PAGE_TRANSFERS is propagated to the page manager, and is used to configure the page manager's communication to the page manager's pgsvr on the new remote cache.

When a Gem triggers creation of a mid-level cache via the method **`midLevelCacheConnect:cacheSizeKB:maxSessions:`**, that Gem's current runtime value of GEM_PGSVR_COMPRESS_PAGE_TRANSFERS is propagated to the page manager, and is used to configure the page manager's communication to the page manager's pgsvr on the new mid-level cache.

Runtime equivalent: **#GemPgsvrCompressPageTransfers**
Default: FALSE

GEM_PGSVR_FREE_FRAME_CACHE_SIZE

GEM_PGSVR_FREE_FRAME_CACHE_SIZE specifies the size of the free frame cache used by the Gem's remote page server. This configuration option has no effect for Gems that are local to the repository extents (which have a page server).

When using the free frame cache, the page server removes enough frames from the free frame list to refill the cache in a single operation. When adding frames to the free list, the page server does not add them until the cache is full.

A value of 0 disables the free frame cache (the page server acquires frames one at a time). A value of –1 means use the default value: 0 for caches less than 100 MB and 10 for caches of 100 MB or greater.

Cache Statistic: FreeFrameCacheSize (Page Server)
Units: frames
Min: –1
Max: 63
Default: –1 (cache size=0 for caches less than 100 MB and 10 for caches of 100 MB or greater)

GEM_PGSVR_FREE_FRAME_LIMIT

GEM_PGSVR_FREE_FRAME_LIMIT determines the free frame limit used by the Gem's remote page server. It has no effect for Gems local to the repository extents (which do not have a page server). For a description of free frames, see the GEM_FREE_FRAME_LIMIT configuration option (page 363).

If the value of GEM_PGSVR_FREE_FRAME_LIMIT is –1, the free frame limit is set to one of the following default values:

- For primary shared page cache that is 800 MB or smaller: 10% of the number of frames in the cache
- For primary shared page cache greater than 800 MB: 5000 frames

To tune the free frame limit of a page server at runtime, use the method `System class>>changeCacheSlotFreeFrameLimit:` *aSlot* `to:` *aValue.*

Cache Statistic: (Page Server) FreeFrameLimit
Min: –1
Max: 65536
Default: –1 (see above discussion)

GEM_PGSVR_UPDATE_CACHE_ON_READ

GEM_PGSVR_UPDATE_CACHE_ON_READ determines the read behavior of the Gem's remote page server when pages are read from disk. If this option is set to true, pages read from disk are also added to the shared page cache on the page server's host. If this option is false, pages read are not added to the page server's shared cache.

This option has no effect for Gems that are local to the repository extents, which do not have page servers, nor on mid-level caches.

Runtime parameter: **#GemPgsvrUpdateCacheOnRead**
Default: false

GEM_PRIVATE_PAGE_CACHE_KB

GEM_PRIVATE_PAGE_CACHE_KB sets the size (in KB) of the Gem's private page cache. This setting also applies to linked Gems.

Min: 128
Max: 524288
Default: 1000

GEM_RPCGCI_TIMEOUT

GEM_RPCGCI_TIMEOUT specifies the time in minutes after which lack of an Rpc command will cause a Gem to terminate. Negative timeouts are not allowed. Resolution of timeouts is one-half the specified timeout interval.

Min: 0
Default: 0 (Gem waits forever)

GEM_RPC_KEEPALIVE_INTERVAL

GEM_RPC_KEEPALIVE_INTERVAL is the interval in seconds for the RPC GCI client keep-alive packet to be sent on the seldom used out-of-band (OOB) socket between the Gem and the GCI client. Has no effect for linked sessions or RPC sessions running the same host as the Gem process.

If enabled, keep-alive packets are sent during the GciPollForSignal() call.

Min: 0
Max: 7200
Default: 0 (disabled)

GEM_SOFTREF_CLEANUP_PERCENT_MEM

GEM_SOFTREF_CLEANUP_PERCENT_MEM controls the cleanup of SoftReferences.

If *startingMemUsed* — the percentage of temporary object memory in-use at the beginning of a VM mark/sweep — is less than the value of this option, no SoftReferences will be cleared.

If *startingMemUsed* is greater than the value of this option and less than 80%, the VM mark/sweep will attempt to clear an internally determined number of least recently used SoftReferences. Under rare circumstances, you might choose to specify a minimum number (GEM_KEEP_MIN_SOFTREFS) that will not be cleared.

If *startingMemUsed* is greater than 80%, VM mark/sweep will attempt to clear all SoftReferences.

Also see the descriptions of the statistics NumSoftRefsCleared, NumLiveSoftRefs, and NumNonNilSoftRefs ("NumLiveSoftRefs (Gem)" on page 499).

Min: 10
Max: 80
Default: 50

GEM_TEMPOBJ_AGGRESSIVE_STUBBING

GEM_TEMPOBJ_AGGRESSIVE_STUBBING controls stubbing in in-memory garbage collection. If instance variable X in object A references object B, and X contains a memory pointer to B, then the reference is *stubbed* by storing into instance variable X the objectId of object B.

When this option is TRUE (the default), references from temporary objects to in-memory copies of committed objects are stubbed whenever possible, during both scavenge and mark/sweep. Also, references from not-dirty in-memory copies of committed objects to other committed objects are stubbed whenever possible. This reduces the number of committed objects forced to stay in-memory, but can slow down garbage collection and subsequent execution.

When this option is FALSE, references from temporary objects to in-memory copies of committed objects are never stubbed. References from not-dirty in-memory copies of committed objects to other committed objects are stubbed after the number of objects flushed during commits reaches a threshold, or if almost OutOfMemory. Performance may be faster, but there is a greater risk of OutOfMemory errors.

Stubbing is always disabled when a commit attempt is in progress, regardless of the setting of this parameter. Certain objects private to the object manager are

always immune from stubbing, and so are references stored into Session State by using `System class >> _sessionStateAt:put:`.

- Also see the descriptions of the statistics NumRefsStubbedMarkSweep and NumRefsStubbedScavenge (page 499).

GEM_TEMPOBJ_CACHE_SIZE

GEM_TEMPOBJ_CACHE_SIZE sets the maximum size (in KB) of the Gem's temporary object memory. (This limit also applies to linked Topaz sessions and linked GemBuilder applications.) This value is set when the VM is initialized and cannot be changed without starting the VM. When you only change this setting, and the other GEM_TEMPOBJ... configuration options use default values, then all of the various spaces remain in proportion to each other.

GEM_TEMPOBJ_CACHE_SIZE defines the maximum memory size. As the actual space required for objects grows, the VM requests and allocates virtual memory as needed.

Cache Statistic: (Gem) GemTempObjCacheSizeKb
Min: 2000
Max: 1000000
Default: 10000

GEM_TEMPOBJ_MESPACE_SIZE

GEM_TEMPOBJ_MESPACE_SIZE sets the maximum size (in KB) of the Map Entries space within the Gem's temporary object memory. This value is set when the VM is initialized and cannot be changed without starting the VM.

One Map Entry is required for each faulted-in committed object, or for any temporary object that might become committed, referenced from an IdentityBag, or exported to the GCI. One Map Entry occupies approximately 24 bytes.

If a Map Entry is needed and the Map Entries Space is full, an OutOfMemory occurs, terminating the session.

Unless you are trying to minimize the memory footprint on HP-UX or AIX, you should always leave GEM_TEMPOBJ_MESPACE_SIZE at its default value (0) so that the system can calculate the size of the Map Entries space based on other memory sizes Otherwise, you are at risk of premature OutOfMemory errors.

Min: 1000
Max: 1000000
Default: 0

GEM_TEMPOBJ_OOPMAP_SIZE

GEM_TEMPOBJ_OOPMAP_SIZE sets the size of the hash table (that is, the number of 8-byte entries) in the objId-to-object map within the Gem's temporary object memory. This value is set when the VM is initialized and cannot be changed without starting the VM.

The value specified is rounded up to the next higher power of 2.

This option should normally be left at its default value (0) so that the system can calculate the size of the map based on other memory sizes.

Min: 16384
Max: 524288000
Default: 0

GEM_TEMPOBJ_POMGEN_PRUNE_ON_VOTE

GEM_TEMPOBJ_POMGEN_PRUNE_ON_VOTE sets the percent of POM generation area to be thrown away when voting on possible dead objects.

If the value is 0, no subspaces of POM generation are cleared; if the value is 100, all subspaces are cleared. For values greater than 0 and less than 100, the number of spaces that are in use and older than 5 minutes is computed, and the specified parameter is the percentage, rounded down, of these subspaces that are discarded.

Runtime equivalent: **#GemPomGenPruneOnVote**
Default: 50
Min: 0
Max: 100

GEM_TEMPOBJ_POMGEN_SCAVENGE_INTERVAL

GEM_TEMPOBJ_POMGEN_SCAVENGE_INTERVAL is the interval in seconds in which the oldest POM generation subspace will be discarded. Lower values may reduce Gem memory usage but may also cause objects to be re-read. Larger values may result in higher Gem memory usage and may reduce disk I/O. Setting this value to zero disables scheduled POM generation scavenges. In this case, POM generation will only be scavenged when all subspaces become full.

Runtime parameter: **#GemTempObjPomgenScavengeInterval**
Units: seconds
Min: 0
Max: 86400
Default: 1800

GEM_TEMPOBJ_POMGEN_SIZE

GEM_TEMPOBJ_POMGEN_SIZE sets the maximum size (in KB) of the POM generation area in the Gem's temporary object memory. This value is set when the VM is initialized and cannot be changed without starting the VM.

The POM generation area holds unmodified copies of committed objects that have been faulted into a Gem, and is divided into ten subspaces.

This option should normally be left at its default value (0) so that the POM generation area is allocated to the default, which is approximately 0.8 times the size of the GEM_TEMPOBJ_CACHE_SIZE.

Min: 1000
Max: 1000000
Default: 0

GEM_TEMPOBJ_SCOPES_SIZE

GEM_TEMPOBJ_SCOPES_SIZE is the size of the scopes stack in Gem temporary object garbage collection. This value is set when the VM is initialized and cannot be changed without starting the VM.

The scopes stack consumes (8 bytes * GEM_TEMPOBJ_SCOPES_SIZE) of C heap memory.

The primary user-visible effect of this setting is maximum depth of nested expressions that can be compiled by the method compiler. The default setting is sufficient for expression nesting of about 200, such as in depth of nested parenthesized expressions.

Min: 1000
Max: 10000000
Default: 2000

KEYFILE

KEYFILE sets the location of GemStone licensing keyfile.

Default: `$GEMSTONE/sys/gemstone.key`

LOG_WARNINGS

If LOG_WARNINGS is set to true, warnings are printed for invalid configuration options.

Default: true

SHR_NUM_FREE_FRAME_SERVERS

SHR_NUM_FREE_FRAME_SERVERS specifies the number of free frame page server processes that will be started when the shared page cache is created.

Min: 0
Max: 30
Default: 1

SHR_PAGE_CACHE_LOCKED

SHR_PAGE_CACHE_LOCKED specifies whether the shared page cache should be locked in memory. On systems that permit a portion of memory to be dedicated to GemStone, this option may provide higher performance.

On Solaris 10, GemStone uses Intimate Shared Memory for the shared page cache, making setting this variable unnecessary.

Other specific operating systems may restrict this action to processes running as root or may require special privileges (such as MLOCK on HP-UX) for this option to take effect; for further information, check the shared page cache monitor log for error messages and consult your operating system documentation.

Default: false

SHR_PAGE_CACHE_NUM_PROCS

SHR_PAGE_CACHE_NUM_PROCS sets the maximum number of processes allowed to attach to the shared page cache. This parameter is used to allocate space in the shared page cache for session information and cache statistics. This cache space is in addition to extent page space allocated by SHR_PAGE_CACHE_SIZE_KB.

The value for SHR_PAGE_CACHE_NUM_PROCS must accommodate the GcGems and various background GemStone processes, as well as user Gem and Topaz session processes. If the value is too small, sessions might be unable to login because they can't attach to the cache. If the value is too large, space in the cache may be wasted.

When the default setting of –1 is specified, the value of this parameter is (STN_MAX_SESSIONS + number of extents in repository + 3).

Cache Statistic: (SPC Monitor) SlotsTotalCount
Min: 15, or the number extents + 3, whichever is larger

Max: determined by STN_MAX_SESSIONS or file descriptor limits
Default: –1

SHR_PAGE_CACHE_NUM_SHARED_COUNTERS

SHR_PAGE_CACHE_NUM_SHARED_COUNTRERS specifies the number of shared counters available in the shared page cache. On most platforms, each counter consumes 128 bytes of shared memory. On AIX, each counter consumes 256 bytes of shared memory. Shared memory used for shared counters is in addition to the shared memory size specified in SHR_PAGE_CACHE_SIZE_KB.

Cache Statistic: (SPC Monitor) NumSharedCounters
Min: 0
Max: 500000
Default: 1900

SHR_PAGE_CACHE_SIZE_KB

SHR_PAGE_CACHE_SIZE_KB sets the size (in KB) of the shared page cache. (Additional shared memory is used for overhead.)

Min: 512
Max: Limited by system memory and kernel configurations
Default: 75000

NOTE

For information about platform-specific limitations on the size of the shared page cache, refer to Chapter 1 of your GemStone/S Installation Guide.

SHR_SPIN_LOCK_COUNT

SHR_SPIN_LOCK_COUNT specifies the number of tries to get a spin lock before the process sleeps on a semaphore. Semaphores involve a relatively time-consuming call to the operating system. Spin locks involve busy–wait loops. Efficient locking may require a combination of these methods.

In single-processor architectures, this value should always be 1 since there is no value in spinning (the lock won't change until the process holding the lock gets scheduled). On multiple-processor architectures, a value of 5000 is recommended.

We recommend that you leave this option set to the default value of –1, which causes GemStone to use a value of either 1 or 5000, based upon the number of CPUs detected.

The runtime parameter can be changed only by SystemUser.

Runtime parameter: **#SpinLockCount**
Default: –1 (use either 1 or 5000, based on the number of CPUs detected)

SHR_TARGET_FREE_FRAME_COUNT

SHR_TARGET_FREE_FRAME_COUNT specifies the target number of free frames to keep in the shared cache at all times. The free frame page server process(es) will attempt to keep the number of free frames in the cache equal to or greater than this value.

If the value of the parameter is –1, the target free frame count is set to a percentage of the total frames in the shared cache. For the main shared cache (the cache to which the stone attaches), the default is 1/8 the frames in the cache. For remote caches, the default is 1/100 the frames in the cache.

For best performance, keep this setting greater than GEM_FREE_FRAME_LIMIT.

If the value of SHR_TARGET_FREE_FRAME_COUNT is –1, the target number of free frames is set to one of the following default values:

- For primary shared page cache that is 800 MB or smaller: 12.5% of the number of frames in the cache
- For primary shared page cache greater than 800 MB: 7000 frames
- For a remote shared page cache: 1% of the number of frames, or 2000 frames, whichever is smaller

Min: –1
Max: 65536
Default: –1 (see above discussion)

SHR_WELL_KNOWN_PORT_NUMBER

SHR_WELL_KNOWN_PORT_NUMBER specifies the port number that the shared page cache monitor will use as its well-known port. The well-known port is used by all Gems and page servers on this host to connect to the cache monitor.

If the specified port is in use by another process, the monitor process will not start and exits with an error.

A value of zero indicates that the port number will be selected by the system.

Min: 1
Max: 65535
Default: 0

STN_ADMIN_GC_SESSION_ENABLED

STN_ADMIN_GC_SESSION_ENABLED determines whether the Admin GcGem is started when the Stone is started. (The Admin GcGem performs administrative garbage collection functions such as write set union sweeps; the Reclaim GcGems perform dead object and page reclamation.)

Runtime parameter: **#StnAdminGcSessionEnabled**
Default: true

STN_ALLOCATE_HIGH_OOPS

If true, the Stone skips the first 16 million object identifiers and begins to allocate object identifiers (GCI OopTypes) for non-special objects at 16r100000001.

This option is designed for testing conversion of GCI applications and user actions. Do not set this option in a production environment.

Default: false

STN_CACHE_WARMER

Specifies if the cache warmer should be run when the Stone is started and whether to load just the object table pages or both the object table and the data pages.

STN_CACHE_WARMER has the following possible values:
0 - Disabled, the stone does not start the cache warmer. This is the default.
1 - Start the cache warmer and load only the object table pages.
2 - Start the cache warmer and load the object table and data pages.

Default: 0

STN_CACHE_WARMER_SESSIONS

Specifies the number of worker sessions (threads) to use by the cache warmer Gem to perform cache warming on startup. In addition to the specified number of threads, there is one additional "master" session allocated. The warmer will exit with an error if not enough sessions are available.

Cache Statistic: NumCacheWarmers (Stone)
Units: sessions

Min: 0
Max: 256
Default: 0 - the warmer is run with numberOfCpus + numberOfExtents sessions

STN_CHECKPOINT_INTERVAL

STN_CHECKPOINT_INTERVAL sets the maximum interval between checkpoints. Checkpoints may be written more often, depending on other factors. The unit is seconds.

This can be changed at runtime only by SystemUser.

Runtime parameter: **#StnCheckpointInterval**
Units: seconds
Min: 5
Max: 1800
Default: 300

STN_COMMIT_QUEUE_THRESHOLD

STN_COMMIT_QUEUE_THRESHOLD determines whether the Stone defers the disposal of commit records, based on the number of sessions in the commit queue. If the size of the commit queue exceeds this threshold, the Stone defers commit record disposal until the commit queue is less than or equal to the value.

This setting is ignored if the commit record backlog exceeds the value of STN_CR_BACKLOG_THRESHOLD.

Runtime parameter: **#StnCommitQueueThreshold**
Default: -1 (never defer commit record disposal)
Min: -1
Max: 1024

STN_COMMIT_RECORD_QUEUE_SIZE

STN_COMMIT_RECORD_QUEUE_SIZE determines the size of the Stone's internal commit record cache. The Stone will keep copies of up to this many commit records in heap memory. Stone is able to dispose commit records more quickly when a copy of the commit record is found in this cache.

When the default value of -1 is specified, Stone sets this value to be twice the value of the STN_SIGNAL_ABORT_CR_BACKLOG option.

Units: Commit Record Pages

Default: -1
Min: 16
Max: 1000000

STN_COMMIT_TOKEN_TIMEOUT

STN_COMMIT_TOKEN_TIMEOUT sets the maximum interval (in seconds) that a session may possess the commit token. If the session possesses the token for longer than this period, the session will be logged off the system and an error message written to the Stone log. GcGems of all types are exempted from this timeout.

Default: 0 (Stone waits forever)
Min: 0
Max: 86400

STN_COMMITS_ASYNC

If STN_COMMITS_ASYNC is set to TRUE, it causes the stone to acknowledge each commit or persistent shared counter update to the requesting session without waiting for the tranlog writes for that commit to complete.

Default: FALSE

STN_CR_BACKLOG_THRESHOLD

STN_CR_BACKLOG_THRESHOLD sets the size of the commit record backlog above which the Stone aggressively disposes of commit records. This setting overrides the deferral of commit record disposal provided by the STN_COMMIT_QUEUE_THRESHOLD parameter.

The default setting of -1 causes the Stone to use a setting equal to (2 * STN_MAX_SESSIONS). A setting of 0 disables this threshold.

Runtime parameter: **#StnCrBacklogThreshold**
Default: -1 (Stone waits forever)
Min: -1
Max: 500000

STN_DISABLE_LOGIN_FAILURE_LIMIT
STN_DISABLE_LOGIN_FAILURE_TIME_LIMIT

These options control when a user account is disabled because the user exceeded the STN_DISABLE_LOGIN_FAILURE_LIMIT of failed login attempts within the time in minutes specified by STN_DISABLE_LOGIN_FAILURE_TIME_LIMIT. When an

account exceeds these limits, the user account is disabled (the system changes the password on the account to one that is invalid) and a record of the event is written to the Stone log file. The user account can only be restored by another user with OtherPassword privileges.

Changes to the runtime parameters require the OtherPassword privilege.

STN_DISABLE_LOGIN_FAILURE_LIMIT:
Runtime parameter: **#StnDisableLoginFailureLimit**
Units: login attempts
Default: 15
Min: 0
Max: 65536

STN_LOG_LOGIN_FAILURE_TIME_LIMIT:
Runtime parameter: **#StnDisableLoginFailureTimeLimit**
Units: Minutes
Default: 15
Min: 1
Max: 1440 (24 hours)

STN_DISKFULL_TERMINATION_INTERVAL

STN_DISKFULL_TERMINATION_INTERVAL specifies how soon (in minutes) the Stone should start terminating sessions holding on to the oldest commit record when the repository free space is below the value set for STN_FREE_SPACE_THRESHOLD. Such sessions are sent the fatal diskfull error.

The runtime parameter can be changed only by SystemUser.

Runtime parameter: **#StnDiskFullTerminationInterval**
Units: Minutes
Min: 0 (no sessions are terminated)
Max: 1440 (24 hours)
Default: 3

STN_EPOCH_GC_ENABLED

STN_EPOCH_GC_ENABLED determines if epoch garbage collection can be run on the system. Leave this value set to FALSE unless you plan to run epoch garbage collection on the system. Setting this to TRUE adds a small amount of overhead to commit processing.

Runtime parameter: **#StnEpochGcEnabled**
Default: FALSE

STN_EXTENT_IO_FLAGS

STN_EXTENT_IO_FLAGS specifies what (if any) special I/O flags will be used to open the database extents. Two kinds of special I/O are supported: direct I/O and concurrent I/O.

Direct I/O tells the operating system avoid caching extent data in the file system cache. Enabling direct I/O tells the operating system to treats the database extents as if they were on raw partitions. Direct I/O may greatly improve database performance in some cases. Concurrent I/O is only available to extents running on the Enhanced JFS file system (aka JFS2) on AIX. Setting this flag has no effect on other operating systems.

STN_EXTENT_IO_FLAGS has the following possible values:

0 - no special I/O flags. This is the default.
1 - enable Direct I/O on all extents on file systems.
2 - enable concurrent I/O (AIX only)

If the requested I/O mode is not available, the stone will fail to start and an error message will be printed in the stone log. If this happens, change this option back to zero and restart the stone.

STN_EXTENT_IO_FLAGS has no effect on extents which reside on raw partitions.

Once the stone starts, all processes which open the database extents (gems and page servers) will open the extents using the same I/O flags. This behavior is required by some operating systems.

Default: 0
Min: 0
Max: 2 (AIX only), 1 (All Others)

STN_FREE_FRAME_CACHE_SIZE

STN_FREE_FRAME_CACHE_SIZE specifies the size of the Stone's free frame cache. When using the free frame cache, the Stone removes enough frames from the free frame list to refill the cache in a single operation.

Units: frames
Default: 1 (disables the free frame cache; Stone acquires frames one at a time)
Min: 1
Max: 1% of the frames in the cache

STN_FREE_SPACE_THRESHOLD

STN_FREE_SPACE_THRESHOLD sets the minimum amount of free space (in MB) to be available in the repository. If the Stone cannot maintain this level by growing an extent, it begins actions to prevent shutdown of the system; for information, see "Repository Full" on page 240.

The runtime parameter can be changed only by SystemUser.

Runtime parameter: **#StnFreeSpaceThreshold**
Default: 1
Min: 0 (no threshold)
Max: 65536

STN_GEM_ABORT_TIMEOUT

STN_GEM_ABORT_TIMEOUT sets the time in minutes that the Stone will wait for a Gem running outside of a transaction to abort (in order to release a commit record), after Stone has signaled that Gem to do so. If the time expires before the Gem aborts, the Stone sends the Gem the error ABORT_ERR_LOST_OT_ROOT, and then either stop the Gem or force it to completely reinitialize its object caches, depending on the value of the related configuration parameter STN_GEM_LOSTOT_TIMEOUT. Negative timeouts are not allowed. Resolution of timeouts is one half the specified time-out interval.

The runtime parameter can be changed only by SystemUser.

Runtime parameter: **#StnGemAbortTimeout**
Default: 1
Min: 1
Max: 1440

STN_GEM_LOSTOT_TIMEOUT

STN_GEM_LOSTOT_TIMEOUT sets the time in seconds that the Stone will wait after signaling the Exception `RepositoryViewLost`, before retracting the Gem's commit record and forcibly stopping the session. Negative timeouts other than –1 are not allowed. Resolution of timeouts is one half the specified timeout interval.

If the value is –1, the Stone does not stop the Gem; it immediately retracts the session's commit record, forcing the Gem to completely reinitialize its object caches.

CAUTION

A value of –1 entails a slight risk that the sleeping Gem will reactivate and begin writing to the shared page cache before it responds to the ABORT_ERR_LOST_OT_ROOT *error, thus corrupting the shared page cache.*

Because of these risks, design your application to minimize the chances of receiving the ABORT_ERR_LOST_OT_ROOT *error.*

The runtime parameter can be changed only by SystemUser.

Runtime parameter: **#StnGemLostOtTimeout**
Default: 60
Min: –1
Max: 5000000

STN_GEM_TIMEOUT

STN_GEM_TIMEOUT sets the time in minutes after which lack of interaction with the Gem causes the Stone to terminate the session. Negative timeouts are not allowed. Resolution of timeouts is one half the specified time-out interval. If non-zero, this timeout is also the maximum time allowed for a Gem to complete processing of its login to the Stone. If this timeout is 0, the maximum time for login processing is set to five minutes.

The runtime parameter can be changed only by SystemUser.

Runtime parameter: **#StnGemTimeout**
Min: 0
Default: 0 (Stone waits forever)

STN_HALT_ON_FATAL_ERR

If STN_HALT_ON_FATAL_ERR is set to true, the Stone will halt and dump core if it receives notification from a Gem that the Gem died with a fatal error that would cause Gem to dump core. By stopping the Stone at this point, the possibility of repository corruption is minimized. true is the recommended setting for systems during development.

If STN_HALT_ON_FATAL_ERR is set to false, the Stone will attempt to keep running if a Gem encounters a fatal error. false is the recommended setting for systems in production use.

Internally, the setting 0 = false, 1 = true

The runtime parameter can be changed only by SystemUser.

Runtime parameter: **#StnHaltOnFatalErr**
Default: false

STN_LOG_IO_FLAGS

STN_LOG_IO_FLAGS specifies what (if any) special I/O flags will be used to open the database transaction logs. Two kinds of special I/O are supported: direct I/O and concurrent I/O.

Direct I/O tells the operating system avoid caching extent data in the file system cache. Enabling direct I/O tells the operating system to treats the database tranlogs as if they were on raw partitions. Direct I/O may greatly improve database performance in some cases.

Concurrent I/O is only available to tranlogs running on the Enhanced JFS file system (aka JFS2) on AIX. Setting this flag has no effect on other operating systems.

STN_LOG_IO_FLAGS has the following possible values:

0 - no special I/O flags. This is the default.
1 - enable Direct I/O on all tranlogs on file systems.
2 - enable concurrent I/O (AIX only)

If the requested I/O mode is not available, the stone will fail to start and an error message will be printed in the stone log. If this happens, change this option back to zero and restart the stone.

STN_LOG_IO_FLAGS has no effect on tranlogs which reside on raw partitions.

Min: 0
Max: 2 (AIX only), 1 (All Others)
Default: 0

STN_LOG_LOGIN_FAILURE_LIMIT
STN_LOG_LOGIN_FAILURE_TIME_LIMIT

If a user has greater than or equal to the STN_LOG_LOGIN_FAILURE_LIMIT number of login failures within the time in minutes specified by STN_LOG_LOGIN_FAILURE_TIME_LIMIT, a message is written to the Stone log file.

Changes to the runtime parameters require the OtherPassword privilege.

STN_LOG_LOGIN_FAILURE_LIMIT:
Runtime parameter: **#StnLogLoginFailureLimit**
Units: login attempts
Min: 0
Max: 65536
Default: 10

STN_LOG_LOGIN_FAILURE_TIME_LIMIT:
Runtime parameter: **#StnLogLoginFailureTimeLimit**
Units: Minutes
Min: 1
Max: 1440 (24 hours)
Default: 10

STN_LOOP_NO_WORK_THRESHOLD

STN_LOOP_NO_WORK_THRESHOLD indicates the maximum number of times the stone will continue executing its main service loop when there is no work to do. If the stone loops more than this number of times and finds no work, the stone will sleep for up to one second. The stone will immediately wake up when there is any work to be done.

Setting this value to zero disables this feature. Setting this value to a non-zero setting, in addition to causing the above behavior, will also cause the stone to not sleep whenever any of the following conditions are true and the no work threshold has not been exceeded:

- a session holds the commit token
- one or more sessions are waiting in the commit queue
- one or more sessions are waiting in the run queue.

Setting this parameter to a non-zero value will always cause the stone to consume more CPU.

Runtime parameter: **#StnLoopNoWorkThreshold**
Default: 0
Min: 0
Max: 536870911

STN_MAX_AIO_RATE

STN_MAX_AIO_RATE specifies the maximum I/O rate that each AIO page server is allowed when performing asynchronous writes. Since the I/O rate specified is applied to each page server, the total maximum I/O rate on the disk system is this value multiplied by STN_NUM_LOCAL_AIO_SERVERS.

The page servers use this maximum I/O rate for both dirty page and checkpoint writes.

Runtime parameter: **#StnMntMaxAioRate**
Min: 20
Max: 1000000
Default: 300

STN_MAX_AIO_REQUESTS

STN_MAX_SESSIONS specifies the maximum number of asynchronous write requests the stone can have pending. If more than this number of asynchronous writes are requested, the stone will wait (sleep) until one or more of the pending requests have completed. Asynchronous write requests are only used to write to the current tranlog.

The maximum value allowed depends on the maximum allowed by the UNIX kernel. The maximum value for this parameter allowed by GemStone is the value of _SC_AIO_MAX or 4096, whichever is lower. On some systems (such as Solaris), it is not possible to determine the value of _SC_AIO_MAX. In that case, GemStone will impose a maximum value of 128. Otherwise the maximum is 4096 or _SC_AIO_MAX, which ever is lower.

For further information on the _SC_AIO_MAX kernel parameter, refer to the UNIX documentation for your system and/or the UNIX man page for the sysconf() call.

Min: 100
Max: the minimum of 4096 or _SC_AIO_MAX; or 128 (see above)
Default: 128

STN_MAX_REMOTE_CACHES

STN_MAX_REMOTE_CACHES specifies the maximum number of remote shared page caches that the system may have.

Min: 0
Max: 10000
Default: 255

STN_MAX_SESSIONS

STN_MAX_SESSIONS limits the number of simultaneous sessions (number of Gem logins to Stone), excluding SymbolGem and GcGems. The actual value used by Stone is the value of this parameter or the number of sessions specified by the software license key file, whichever is less. This parameter is provided so the number of users to be restricted to avoid overloading the host computer. The maximum number of file descriptors per process (imposed by the operating system kernel) can also limit the maximum number of sessions.

If you increase STN_MAX_SESSIONS, and you are not allowing the system to calculate SHR_PAGE_CACHE_NUM_PROCS, you will need to increase that parameters as well.

Recommended: 40 or more, depending on your requirements
Min: 1
Max: 10000
Default: 40

STN_MAX_VOTING_SESSIONS

STN_MAX_VOTING_SESSIONS specifies the maximum number of sessions that can simultaneously vote on possible dead objects, at the end of a markForCollection or epoch garbage collection. To help prevent the voting on possible dead objects from causing large increases in response time of the system, set this to a value substantially lower than STN_MAX_SESSIONS.

Runtime parameter: **#StnMaxVotingSessions**
Min: 1
Max: 1000000
Default: 100

STN_NUM_AIO_WRITE_THREADS

STN_NUM_AIO_WRITE_THREADS specifies the number of native threads the Stone will start to perform writes to the tranlog. In commit-intensive systems, this should be increased to 8 or 16.

Cache Statistic: StnAioNumWriteThreads (Stone)
Min: 4
Max: 256
Default: 4

STN_NUM_GC_RECLAIM_SESSIONS

STN_NUM_GC_RECLAIM_SESSIONS specifies the number of page reclaim garbage collector gems (Reclaim GcGems) that will be started when the Stone starts. The maximum number of Reclaim GcGems is equal to the number of extents as specified in DBF_EXTENT_NAMES. If you specify a value larger than the number of extents, one GC reclaim will be started for each extent.

Runtime parameter: **#StnNumGcReclaimSessions**
Default: 1
Min: 0
Max: 256

STN_NUM_LOCAL_AIO_SERVERS

STN_NUM_LOCAL_AIO_SERVERS is the approximate number of page server processes to start as local asynchronous I/O servers for the shared page cache on the node where the Stone runs. The number of extents known to the Stone at startup is divided by the value of STN_NUM_LOCAL_AIO_SERVERS to compute the internal configuration parameter `StnRDbfMaxFilesPerServer`. The latter parameter is the approximate number of extent files to be serviced by each AIO page server.

For instance, if your configuration has six extents, setting STN_NUM_LOCAL_AIO_SERVERS to 3 causes each AIO page server to service (6 ÷ 3) = 2 extents.

Under certain circumstances, multiple AIO page servers can help you achieve the maximum possible commit rate. A value greater than one is recommended only if there are two or more extents, the host has multiple CPUs (to allow parallel

execution), and the disk drive hardware allows concurrent writes to disk (the extents are on separate spindles, or the equivalent).

Min: 1
Max: 256
Default: 1

STN_OBJ_LOCK_TIMEOUT

STN_OBJ_LOCK_TIMEOUT specifies the time in seconds that a session is allowed to wait to obtain one of the special single object write locks. For more information, see the methods System >> waitForRcWriteLock: and System >> waitForApplicationWriteLock:queue:autoRelease:.

Runtime parameter: **#StnObjLockTimeout**
Min: 0
Max: 86400
Default: 0 (stone waits forever)

STN_PAGE_MGR_COMPRESSION_ENABLED

STN_PAGE_MGR_COMPRESSION_ENABLED determines if the page manager will compress the list of pages that it sends to remote shared page caches for removal. If set to TRUE, all lists of pages larger than 50 will be compressed before transmission. The same compressed list is used to send to all remote shared page caches; i.e., the compression operation is performed no more than once for each list of pages to be sent. Has no effect on systems that do not use remote shared page caches.

Runtime equivalent: **#StnPageMgrCompressionEnabled**
Cache Statistic: PageMgrCompressionEnabled (Stone)
Default: FALSE

STN_PAGE_MGR_PRINT_TIMEOUT_THRESHOLD

STN_PAGE_MGR_PRINT_TIMEOUT_THRESHOLD is the threshold in real seconds used by the page manager to determine if a slow response from a remote shared page cache should be printed to the page manager log file. If a remote cache takes longer than this number of seconds to respond to the page manager, the page manager will print a message to the log file. If a remote cache takes less than this number of seconds to respond, no message is printed.

Note that this value controls the writing of log messages only. The connection to the remote cache will not be terminated by page manager unless STN_REMOTE_CACHE_PGSVR_TIMEOUT is exceeded.

Runtime equivalent: **#StnPageMgrPrintTimeoutThreshold**
Cache Statistic: PageMgrPrintTimeoutThreshold (Stone)
Min: 0
Max: 3600
Default: 5

STN_PAGE_MGR_REMOVE_MAX_PAGES

STN_PAGE_MGR_REMOVE_MAX_PAGES sets the maximum batch size for the Page Manager gem. This is the maximum number of pages in a single request to the stone. Must be greater than or equal to STN_PAGE_MGR_REMOVE_MIN_PAGES

Runtime parameter: **#StnPageMgrRemoveMaxPages**
Cache Statistic: PageMgrRemoveMaxPages (Stone)
Min: 1
Max: 16384
Default: 16384

STN_PAGE_MGR_REMOVE_MIN_PAGES

STN_PAGE_MGR_REMOVE_MIN_PAGES sets the minimum batch size for the Page Manager gem. When the number of pages waiting to be processed by the Page Manager is greater than this value, then the Page Manager will request the pages from the stone and process them. Otherwise the Page Manager will wait until this threshold is exceeded before requesting pages from the stone. Must be less than or equal to STN_PAGE_MGR_REMOVE_MAX_PAGES

Runtime parameter: **#StnPageMgrRemoveMinPages**
Cache Statistic: PageMgrRemoveMinPages
Min: 0
Max: 1792
Default: 40

STN_PRIVATE_PAGE_CACHE_KB

STN_PRIVATE_PAGE_CACHE_KB sets the default size (in KB) of the Stone page cache.

Min: 128
Max: 524288
Default: 2000

STN_REMOTE_CACHE_PGSVR_TIMEOUT

STN_REMOTE_CACHE_PGSVR_TIMEOUT specifies the maximum time in seconds that the page manager session will wait for a response from a page server on a remote shared page cache. If no response is received within the timeout period, all Gems attached to that cache are logged off and a message is written to the Stone and page manager logs. Negative timeouts are not allowed. A timeout value of zero causes the page manager to wait forever.

Runtime equivalent: **#StnRemoteCachePgsvrTimeout**
Cache Statistic: PageMgrRemoteCachePgsvrTimeout (Stone)
Min: 0
Max: 3600
Default: 15

STN_REMOTE_CACHE_TIMEOUT

STN_REMOTE_CACHE_TIMEOUT sets the time in minutes after the last active process on a remote node logs out before the Stone shuts down the shared page cache on that node.

A value of 0 causes the Stone to shut down the remote cache as soon as possible.

The runtime parameter can be changed only by SystemUser.

Runtime parameter: **#StnRemoteCacheTimeout**
Min: 0
Max: 5000000
Default: 5

STN_SHR_TARGET_PERCENT_DIRTY

STN_SHR_TARGET_PERCENT_DIRTY specifies the maximum percentage of the Stone's shared page cache that can contain dirty pages without AIO page servers increasing their IO rates.

Runtime parameter: **#StnShrPcTargetPercentDirty**
Min: 1
Max: 90
Default: 20

STN_SIGNAL_ABORT_CR_BACKLOG

STN_SIGNAL_ABORT_CR_BACKLOG sets the number of old transactions (commit records) above which the Stone will start to generate SignalAbort messages to a Gem that is running outside of a transaction. You may need to tune this option according to your application's commit rate and your system's tolerance for the possible swapping activity caused by awakening sleeping session processes.

The runtime parameter can be changed only by SystemUser.

Runtime parameter: **#StnSignalAbortCrBacklog**
Default: 20
Min: 0
Max: 65536

STN_TRAN_FULL_LOGGING

If STN_TRAN_FULL_LOGGING is set to true, all transactions are logged, and log files are not deleted by the system. In this mode, the transaction logs are providing real-time incremental backup of the repository. If no disk space is available for logs, Gem session processes may appear to "hang" until space becomes available.

If STN_TRAN_FULL_LOGGING is set to false, only transactions smaller than STN_TRAN_LOG_LIMIT are logged; larger transactions cause a checkpoint, which updates the extent files. Log files are deleted by the system when the circular list of log directories wraps around. This setting allows a simple installation to run unattended for extended periods of time, but it does **not** provide real-time backup. See also STN_TRAN_LOG_DEBUG_LEVEL, which can cause old log files to be retained under partial logging.

Once you have started the Stone on a repository with STN_TRAN_FULL_LOGGING = true, then the true state will persist in the repository; any subsequent changes to this parameter in the configuration file are ignored. To change the repository back to partial logging, you must do a Smalltalk full backup and then restore the backup into a copy of `$GEMSTONE/bin/extent0.dbf`.

For further information, see Chapter 8, "Managing Transaction Logs."

Default: None. The system will not run unless you provide a value.
Initial setting: false

STN_TRAN_LOG_DEBUG_LEVEL

This option is only for GemStone internal use. Customers should not change the default setting unless directed to do so by GemStone Technical Support. Values ≥ 2 inhibit removal of old transaction logs when in partial logging mode.

Runtime parameter: **#StnTranLogDebugLevel**
Default: 0

STN_TRAN_LOG_DIRECTORIES

STN_TRAN_LOG_DIRECTORIES lists the directories or raw disk partitions to be used for transaction logging. This list defines the maximum number of log files that will be online at once. Each entry must be a directory or a raw disk partition. Directories may appear multiple times in the list. A given raw disk partition may appear only once.

Min: 2 entries
Max: 2^{31} entries
Default: Empty (the system will not run without at least two entries)
Initial setting: `$GEMSTONE/data/, $GEMSTONE/data/`

STN_TRAN_LOG_LIMIT

STN_TRAN_LOG_LIMIT sets the maximum transaction log entry size limit in KB. Successful commits of transactions consuming more than this amount of log file space when STN_TRAN_FULL_LOGGING is set to false will cause a checkpoint. This option has no effect when STN_TRAN_FULL_LOGGING is set to true.

The runtime parameter can be changed only by SystemUser.

Runtime parameter: **#StnTranLogLimit**
Min: 25
Max: 1000
Default: 1000

STN_TRAN_LOG_PREFIX

STN_TRAN_LOG_PREFIX sets file name prefixes for transaction log files. A sequence number and "`.dbf`" is added to the prefix; for example, "`tranlog`" produces "`tranlog0.dbf, tranlog1.dbf, ...`". You can set this configuration option to permit multiple repository monitors to share a log directory without conflict.

Default: `tranlog`

STN_TRAN_LOG_SIZES

STN_TRAN_LOG_SIZES sets the maximum sizes (in MB) of all log files, in order and separated by commas. Each size applies to a corresponding log file specified in STN_TRAN_LOG_DIRECTORIES, and the number of entries must match. The sizes also apply to the corresponding entries in STN_REPL_TRAN_LOG_DIRECTORIES when that list is not empty.

For transaction logs on raw partitions, if the size specified is larger than the physical size of the corresponding raw partition, the STN_TRAN_LOG_SIZES value is adjusted appropriately.

Min: 3
Max: 16384
Default: Empty (the system will not run unless sizes are specified)
Initial setting: 100, 100

STN_TRAN_Q_TO_RUN_Q_THRESHOLD

STN_TRAN_Q_TO_RUN_Q_THRESHOLD specifies the number of sessions in the commit queue (waiting for the commit token) above which the stone will allow the remaining sessions in the queue to process unions (read old commit records) while waiting for the commit token.

For example, if this parameter is set to 6 and there are 9 sessions in the commit queue, the 3 last sessions will be allowed to process unions while waiting for the token. If there are 6 or fewer sessions in the queue, no sessions will process unions.

The first session in the commit queue never processes unions since it will receive the token when the current commit completes.

Runtime parameter: **#StnTranQToRunQThreshold**
Min: 1
Max: 1024
Default: 6

STN_WELL_KNOWN_PORT_NUMBER

STN_WELL_KNOWN_PORT_NUMBER is the port number that the Stone will use as its well-known port. The well-known port is used by all Gems while establishing their initial connection to the Stone during the login sequence.

If the specified port is in use by another process, the Stone will not start and exits with an error.

A value of zero indicates the port number will be selected by the system.

Min: 1
Max: 65535
Default: 0

A.4 Runtime-only Configuration Parameters

The parameters described in this section are similar to the configuration parameters above, but can only be read or modified at runtime.

The process for modifying is similar to that for the runtime parameter equivalents for the configuration parameters listed in the configuration files.

The runtime parameters are read using the method `System class>>configurationAt:`, and updating using `System class>>configurationAt:put:`.

#GemConvertArrayBuilder

If True, allows old style Array Builder syntax #[a, b] to be parsed correctly. The compiler converts this syntax to the new form { a . b }, and updates method source as well as compiled code.

Default: false

#GemDropCommittedExportedObjs

If this configuration parameter is true, clean, committed objects may be dropped from RAM. This reduces demand on memory in the Gem, but there is the small cost of an additional bitmap lookup when the object is faulted, to detect if this object is in the Pure Export Set.

Default: false

#GemExceptionSignalCapturesStack

If this configuration parameter is true, invocations of AbstractException>>_signalWith: fill in the gsStack instance variable of the receiver, allowing subsequent calls to Exception >> stackReport.

Default: false

#LogOriginTime

`#LogOriginTime` is the time the current sequence of Stone logs was started. It is the same value returned by `Repository>>logOriginTime`. For information about when a new sequence is started, see the method comment for `Repository>>commitRestore` in the image.

This should not be modified.

#SessionInBackup

`#SessionInBackup` is the GemStone session number of the session performing a full backup, or –1 if a backup is not in progress.

This should not be modified by the user.

#StnCurrentTranLogDirId

`#StnCurrentTranLogDirId` is the one-based offset of the current transaction log into the list of log directory names, STN_TRAN_LOG_DIRECTORIES. It is the same value returned by `Repository>>currentLogDirectoryId`.

This should not be modified.

#StnCurrentTranLogNames

`#StnCurrentTranLogNames` is an Array containing up to two Strings: the name of the transaction log to which records currently are being appended, and the name of the current replicated log. These are the same values returned by `Repository>>currentLogFile` and `currentLogReplicate`, respectively.

#StnLogFileName

The stone log file path and name.

This is a read-only value.

#StnLogGemErrors

`#StnLogGemErrors` is intended for internal debugging use. When it is set to 1, the Stone logs error messages it sends to Gems.

#StnLoginsSuspended

`#StnLoginsSuspended` ordinarily has the values 0 (false) and 1 (true) as set by `System class>>suspendLogins` and `resumeLogins`.

Changing this parameter requires the SystemControl privilege.

#StnMaxReposSize

`#StnMaxReposSize` is the maximum size of the repository for your GemStone license, as set by the GemStone keyfile. The value of this parameter is 0 when not limited by the keyfile.

This should not be modified.

#StnMaxSessions

`#StnMaxSessions` is the maximum allowed number of sessions for your GemStone license, as set by the GemStone keyfile. It is not related to the STN_MAX_SESSIONS configuration option. The value of this parameter is 0 when not limited by the keyfile.

This should not be modified.

#StnSunsetDate

`#StnSunsetDate` is the sunset date for your GemStone license, as set by the GemStone keyfile. The value of this parameter is 0 when not limited by the keyfile.

This should not be modified.

#StnTranLogOriginTime

`#StnTranLogOriginTime` is the time when the current transaction log was started.

This should not be modified.

Appendix

B GemStone Utility Commands

The GemStone/S 64 Bit utility commands in this appendix are provided in the $GEMSTONE/bin directory. For most of these commands, you can display usage information by entering the command name with the **-h** option. For example:

```
% pageaudit -h
Usage: pageaudit [-d] [-h] [-f] [-l logfile] [-e execfg]
       [-z syscfg] [name]
  where -d   disables auditing of data pages.
        -h   prints usage information and exits.
        -f   keeps running beyond first error if possible
...
```

UNIX man pages are available for more detailed information on each command.

copydbf

copydbf *sourceNRS destinationNRS* [**-C**] [**-E**] [**-f***filePrefix*] [**-n***netLdiName*] [**-p***pgsvrId*] [**-s***Mbytes*] [**-h** | **-l** | **-m** | **-P**]

copydbf *sourceNRS* [**-i** | **-I**] [**-n***netLdiName*] [**-p***pgsvrId*]

sourceNRS The source file or raw partition (containing an extent, a transaction log, or a full backup) as a GemStone network resource string.

destinationNRS The destination file, directory, or raw partition as a GemStone network resource string. If the destination is a file system directory (the trailing / is optional), a file name is generated and appended to *destinationNRS* based on the type and internal fileId of the source. Use of `/dev/null` as the destination is supported only for files as a means of verifying that the file is readable.

-C Compress output. The output must be a filesystem file. Write the output compressed, in gzip format. The output file name will have the suffix `.gz` appended to it, if it does not end in `.gz`.

-E Ignore disk read errors. If the disk read error occurs while reading an extent root page, the copy will fail. Otherwise, pages of the source file that cannot be read will be replaced with an invalid-page-kind page in the destination file. The destination file may not be usable.

This option only applies to extents, not to transaction logs or backup files.

-f*filePrefix* If *destinationNRS* is a file system directory, then *filePrefix* overrides the filename prefix that would be generated based on the contents of *sourceNRS*. If *destinationNRS* is other than a file system directory, this option has no effect.

-n*netLdiName* The name of the GemStone network server; the default is `gs64ldi`.

-p*pgsvrId* The name of a specific `runpgsvr` (similar to `gemnetid`).

-s*MB*	The size (in MB) to pre-allocate the destination file. For instance, **-s**10 allocates at least 10 MB to the created file. If you do not specify the **-s** option, the output file is made as short as possible.
-h	Displays a usage line and exits.
-i	Information only. When this option is present without *destinationNRS,* information about *sourceNRS* is printed without performing a file copy. If both **-i** and *destinationNRS* are present, an error message is printed.
-I	Full information. The same information is printed as for **-i.** In addition, if the file is a transaction log, all checkpoint times found are listed instead of only the last one.
-l	Least-significant-byte ordering for the *destinationNRS.* This byte ordering is the native byte ordering for Intel processors.
-m	Most-significant-byte ordering for the *destinationNRS.* This byte ordering is the native byte ordering for Solaris SPARC, AIX POWER, and HP-UX PA-RISC and Itanium processors.
-P	Preserve byte ordering. This option creates the destination file using the byte ordering found in the source file. The default is to write the file using the host's native byte ordering.

The **copydbf** utility requires exclusive access to repository files in order to copy them without corruption. You must shut down the Stone repository monitor before copying an extent. You may copy any transaction log that is not the active log. copydbf -i or -I, which reports information but do not make a copy, can be executed on the active transaction log.

GemStone repository files on the UNIX file system can usually be copied using the ordinary **cp** command. However, if you are copying between operating systems and the source and destination have different byte ordering, you should use **copydbf**. You can use the first form of **copydbf** for disk-to-disk copies between machines (without NFS) or copies to and from raw disk partitions. To use **copydbf** between remote nodes, you must have a NetLDI running on both nodes, and authentication must be configured to allow access.

You must give an NRS (network resource string) for both the source file and the destination. A local machine file specification is a valid NRS.

If the destination is a directory in a file system, **copydbf** generates a filename based on the type of file. The generated name includes a prefix (`extent`, `tranlog`, or

backup), a fileId representing an internal sequence number that starts at 0, and the extension .dbf. You can use the **-f** option to change the prefix.

A message describing the source and destination files is printed to standard error before starting the copy. The size of the destination file is printed to standard error after the copy is completed. For example:

```
% copydbf $GEMSTONE/data/extent0.dbf .

Source file: /users/GemStone/data/extent0.dbf
   file type: extent  fileId: 0
   byteOrder: Sparc (MSB first) compatibilityLevel: 843
   Last checkpoint written at: 01/21/11 16:20:29 PST.
Destination file:  ./extent0.dbf
   byteOrder: Sparc (MSB first)
   Clean shutdown, no tranlog needed for recovery,
   last tranlog written to had fileId 5 ( tranlog5.dbf ).
   File size is 54.5 MBytes (3328 records).
```

To obtain the same source file information (but not the size) without making a copy, use the second form of the command: **copydbf -i** *sourceNRS*. In this usage, you do not specify a *destinationNRS*.

The following **copydbf -i** example displays information for an extent, and indicates the oldest transaction log that would be needed to recover from a system crash:

```
% copydbf -i extent0.dbf

Source file: extent0.dbf
   file type: extent  fileId: 0
   byteOrder: Sparc (MSB first) compatibilityLevel: 843
   Last checkpoint written at: 01/21/11 16:20:29 PST.
   Oldest tranlog needed for recovery is fileId 5
   ( tranlog5.dbf ).
   Extent was shutdown cleanly; no recovery needed.
```

The next example displays information for a backup, and indicates the oldest transaction log that would be needed to restore subsequent transactions.

```
% copydbf -i backup.dat

Source file: backup.dat
   file type: backup  fileId: 0
   byteOrder: Sparc (MSB first)  compatibilityLevel: 843
   The previous file last recordId is  -1.
Destination file:  /dev/null
   Full backup started from checkpoint at: 01/21/11 9:28:37 PST
   Oldest tranlog needed for restore is fileId 5
   ( tranlog5.dbf ).
```

To obtain the size of a repository file in a raw partition, use this form:

```
% copydbf sourceNRS destinationNRS
```

For a listing of all checkpoints recorded in a transaction log, use **copydbf -I** *sourceNRS*. This information is helpful in restoring a GemStone backup to a particular point in time. For example:

```
% copydbf -I tranlog5.dbf

Source file: tranlog5.dbf
   file type: tranlog  fileId: 5
   byteOrder: Sparc (MSB first) compatibilityLevel: 830
   The file was created at: 01/21/11 15:28:18 PST.
   The previous file last recordId is  36 .
   Scanning file to find last checkpoint...
Destination file:  /dev/null
   byteOrder: Sparc (MSB first)
   Checkpoint 1 started at: 01/21/11 15:28:18 PST.
     oldest transaction references fileId -1 ( this file ).
   Checkpoint 2 started at: 01/21/11 15:28:37 PST.
     oldest transaction references fileId -1 ( this file ).
   Checkpoint 3 started at: 01/21/11 15:30:30 PST.
     oldest transaction references fileId -1 ( this file ).
   Checkpoint 4 started at: 01/21/11 15:57:01 PST.
     oldest transaction references fileId -1 ( this file ).
   File size is 35328 bytes (69 records).
```

To pre-allocate disk space in the destination file, use the **-s** option. For instance, **-s50** would allocate at least 50 MB to the created file. The output file is made as short as possible by default.

In the following example, the local GemStone repository file "extent0.dbf" is copied to a remote machine using a full *destinationNRS*. In this example, the repository file is copied to a remote machine named "node," using remote user account "username" and "password," with a remote filespec of "/users/path/extent0.dbf_copy," via the standard GemStone network server `gs64ldi` using TCP protocol:

```
% copydbf $GEMSTONE/data/extent0.dbf \
!@node#auth:username@password#dbf!/users/extent0.dbf_copy
```

The next example copies a fresh repository extent to an existing raw disk partition. If the raw partition already contains a repository file or backup, use **removedbf** first to mark it as being empty.

```
% copydbf $GEMSTONE/bin/extent0.dbf /dev/rsd3h
```

Copying a transaction log from a raw partition to a file system directory generates a file name having the same form as if that transaction log had originated in a file system. If the internal fileId is 43, this example would name the destination file `/dsk1/tranlogs/tranlog43.dbf`:

```
% copydbf /dev/rsd3h /dsk1/tranlogs/
```

gslist

gslist	[**-h** | **-l** | **-p** | **-x**] [**-c**] [**-q**] [**-v**] [**-t** *secs*] [**-u** *user*] [**-m** *host*]+ [[**-n**] *name*]+
-h	Prints a usage line and exits.
-l	Prints a long listing (includes pid and port).
-p	Prints only the pid (process id), or 0 if the server does not exist.
-x	Prints an exhaustive listing, with each item on a separate line.
-c	Removes locks left by servers that have been killed.
-q	(Quiet.) Don't print any extra information; intended for use when the output will be processed by some other program.
-v	Verify the status of each server.
-t *secs*	Wait *secs* seconds for server to respond (only with **-v**); default is 2 seconds.
-u *user*	Only list servers started by *user.*
-m *host*	Only list servers on machine *host*; default is '.' which represents the local host. If not the local host, the machine *host* must be running a compatible version of netldi with the default netldi name for the environment in which gslist is being executed.
[**-n**] *name*	Only list the server *name.*

The **gslist** command prints information about GemStone servers. The default listing prints the following server attributes in columns:

Status	One of the following:	
	OK	server is accepting clients (-v only)
	frozen	server is not responding (-v only)
	full	server can't accept any more clients (-v only)
	exists	server process exists but is not verified
	killed	server process does not exist
	startup	server process is not yet accepting clients
Version	GemStone version of the server.	

Owner	The account name of the user who created the server.
Started	The date and time that the server was started.
Type	One of the following: Netldi, Stone, or cache (shared page cache monitor)
Name	The server's name.

When you include the **-l** or **-x** option, the following additional columns are printed:

Pid	The process id of the server's main process.
Port	The port number of the server's listening socket.

The **-x** option prints the preceding attributes on separate lines, and adds lines for the following as appropriate:

options	Options used when the server was started.
logfile	Full path of server's log file, if it exists.
sysconf	The GemStone system configuration file (see page 350).
execonf	The GemStone executable configuration file (see page 353).
GEMSTONE	Root of the product tree used by the server.

If many servers are reported as **frozen**, try increasing `-t` *secs*.

By default, status is returned for all servers on the current host. To specify a particular server, use the -n switch, or just include the server name on the command line (since the -n is optional). To specify multiple server names, include the **-n** *name* option for each server.

The **-m** option allows you to list servers on a remote host. To specify more than one remote host, include the **-m** *host* option for each host. Names on remote hosts are prefixed by `host:`, where *host* is the name of the remote machine.

To list servers on a remote host, **gslist** must be able to contact a NetLDI running on the remote machine. The remote NetLDI must be named according to the default NetLDI for the environment in which **gslist** is running, either "gs64ldi" or as specified by $GEMSTONE_NRS_ALL. In addition, the GemStone version of the remote NetLDI must be compatible with **gslist**.

The exit status has the following values:

0	Operation was successful.
1	No servers were found.
2	A stale lock was removed (in response to **-c** switch).
3, 4	An error occurred.

pageaudit

pageaudit	[*gemStoneName*] [**-e** *exeConfig*] [**-z** *systemConfig*] [**-f**] [**-d**] [**-l** *logfile*] [**-h**]
gemStoneName	Name of the GemStone repository monitor; the default is `gs64stone-audit`. Network resource syntax is not permitted.
-e *exeConfig*	The GemStone executable-dependent configuration file (see page 353).
-z *systemConfig*	The GemStone system configuration file (see page 350).
-f	Keeps running beyond the first error, if possible.
-d	Disable audit of data pages. Only audit Object Table pages, bitmaps, and other non-data pages.
-l *logfilename*	Write output to the file with the given name. The file is created if it does not exist. If there is an existing file with this name, the pageaudit output is appended to the end of this file.
-h	Displays a usage line and exits.

Audit the pages in a GemStone repository, which must not be in use. **pageaudit** opens the repository specified by the relevant configuration files. The arguments **-e** *exeConfig*, and **-z** *systemConfig* determine which configuration files **pageaudit** reads. Other options for this command generally are not needed.

By default, all pages in the repository are verified; this includes data pages as well as Object Table, bitmap, and other pages containing internal information. Audit of data pages can be disabled using the -d option.

An error is returned if another Stone is running as *gemStoneName* or has opened the same repository.

When you include the **-f** switch, **pageaudit** prints all errors possible. Without **-f**, the default is to stop after the first error is found.

This utility can take a long time to run, so it is best to run it as a background job.

For additional information about **pageaudit** and a description of its output, see "To Perform a Page Audit" on page 165.

removedbf

removedbf	*dbfNRS* [**-n** *netLdiName*] [**-p** *pgsvrNetId*] [**-h**]
dbfNRS	The GemStone repository filename or device, as a network resource string, for the repository to be removed.
-n *netLdiName*	The name of the GemStone network server; default is `gs64ldi`.
-p *pgsvrNetId*	The name of a specific page server to use.
-h	Displays a usage line and exits.

This command removes (erases) a GemStone repository file. It is provided primarily for erasing an extent or transaction log from a raw partition, but it also works on files in the file system and on remote machines. You must provide the NRS (network resource string) of the file to be removed. (A local machine filespec is a subset of an NRS.)

If you specify a file in the file system, this command is equivalent to the **rm** command. If you specify a raw disk partition, GemStone metadata in the partition is overwritten so that GemStone will no longer think there is a repository file on the partition.

This command does not disconnect an extent from the logical repository. To alter the configuration by disconnecting an existing extent, see "How To Remove an Extent" on page 233.

The options for this command generally are not needed for a standard GemStone configuration.

startcachewarmer

startcachewarmer [**-d**|**-D**] [**-h**] [**-l** *limit*] [**-L** *path*] [**-n** *numGems*] [**-p** *password*] [**-s** *stone*][**-T** *int*][**-u** *userID*] [**-w** *delayTime*] [**-W**]

-d Reads data pages into the cache (default: only object table pages are read).

-D Reads data pages into the UNIX file buffer cache and not the shared page cache.

-h Displays a usage line and exits.

-L *path* Path to a writable log file directory (default: current directory)

-l *limit* Stops cache warming if the number of free frames in the cache falls below the specified *limit*. If *limit* is -1 (the default), have the system compute the actual limit based on the size of the shared cache. If *limit* is 0, force cache warming to continue even if the shared cache is full.

-n *numGems* Number of Gem sessions to start (default: 1).

-p *password* GemStone password for logging in Gems (default: `swordfish`).

-s *stone* Name of the running Stone (default: `gs64stone`).

-T *int* Sets the Temporary Object Cache size that will be used by the cache warmer Gems. (default: 5 MB).

-u *userID* GemStone user for logging in Gems (default: `DataCurator`).

-w *delayTime* Wait *delayTime* seconds between spawning Gems (default: 1)

-W Wait for cache warming Gems to exit before exiting this script. By default, this script spawns Gems in the background and exits immediately.

The **startcachewarmer** command warms up the shared page cache on startup, by preloading object and data tables into the cache. This allows the overhead of initial page loading to occur in a controlled way on system startup, rather than more gradually as the repository is in use.

Cache warming runs in one or two phases:

- In the first phase, the object table is loaded into the shared page cache. During this phase, if any data pages will be loaded later, the data pages that are referenced by the object table lookups are recorded for use in phase 2.
- In the second phase, data pages remembered from the first phase are loaded either into the shared page cache or the file buffer. This phase runs only if data pages will be loaded into the shared page cache or the file buffer.

For greater efficiency, you can set the configuration file parameters STN_CACHE_WARMER and STN_CACHE_WARMER_SESSIONS to perform cache warming on startup. See Appendix A for more details on these parameters.

startnetldi

startnetldi [*netLdiName*] [**-l** *logFile*] [**-t** *timeout*] [**-a** *name*] [**-p** *low*:*high*] [**-P** *portNumber*] [**-n**] [**-g** | **-s**] [**-d**] [**-h**]

netLdiName The name or port number of the GemStone network server. If *netLdiName* is a numeric value equal to or less than 65535, it is interpreted as a port number. If the given port is in use, it will result in an error. Other values are interpreted as the netldi name. Network resource syntax is not permitted. The default is `gs64ldi`.

-l *logFile* The logged output of the NetLDI; the default is `/opt/gemstone/log/`*NetLdiName*`.log`

-t *timeout* Seconds to wait for a spawned client to start, such as a Gem; default is 30 seconds.

-a *name* Captive account; all child processes created by the NetLDI will belong to the account named *name*. By default, child processes belong to the client's account.

-**p** *low*:*high* The low and high ports in the pool of ports to use in creating processes.

-**P** *portNumber* The well-known port number that netldi will use.

-n Do not create any *ad hoc* processes (ad hoc processes are ones not listed in `$GEMSTONE/sys/services.dat`).

-**g** Guest mode; no accesses are authenticated. This option is not allowed if the NetLDI's effective user id is the root account.

-s Secure; require authentication for **all** NetLDI accesses.

-d Debug mode; inserts more extensive messages in the log file.

-**h** Displays a usage line and exits.

This command starts a GemStone network server with the specified *netLdiName* and *timeout* (given in seconds). The server spawns GemStone processes in response to login requests from remote applications and requests for remote repository access from Stone repository monitor processes. If your machine is slow or heavily loaded, and RPC logins time out before completing, specify a larger timeout value.

The NetLDI listens for requests, including RPC login requests, on a well-known port, and then uses another set of ports to establish communications with the Stone. If you are running over a firewall, you can specify the ports using the **-p** and **-P** options, and configure your firewall to permit access via these ports.

To start the NetLDI for password authentication, make sure that `$GEMSTONE/sys/netldid` is owned by root and has the S bit set. Issue this command (on some operating systems, you may have to issue it as root):

```
% startnetldi
```

To start the NetLDI in guest mode (authentication is not required), make sure `$GEMSTONE/sys/netldid` does NOT have the S bit set. Log in as the captive account *name*, then issue this command:

```
% startnetldi -g -aname
```

For information about authentication modes, see "How To Arrange Network Security" on page 99.

For assistance with startup failures, refer to "To Troubleshoot NetLDI Startup Failures" on page 135.

startstone

startstone [*gemStoneName*] [**-l** *logFile*] [**-e** *exeConfig*] [**-z** *systemConfig*] [**-h**] [**-R**][**-N]**

gemStoneName Name of the GemStone repository monitor, default is `gs64stone`. Network resource syntax is not permitted.

-l *logFile* The location (or filename) for the logged output of the stone; the default is (1) the setting of the `GEMSTONE_LOG` environment variable, if defined;
(2) `$GEMSTONE/data/`*gemStoneName*`.log`

-e *exeConfig* The GemStone executable-dependent configuration file (see page 353).

-z *systemConfig* The GemStone system configuration file (see page 350).

-h Displays a usage line and exits.

-R Start up from the most recent checkpoint and go into restore-from-transaction-logs state. This option should be used only for recovery using operating system backups.

-**N** Do not replay transaction logs as part of startup. Unless used with the **-R** option, and transaction logs are replayed manually, work may be lost.

This command opens the GemStone repository specified by the relevant configuration files. The three arguments *gemStoneName*, **-e** *exeConfig*, and **-z** *systemConfig* determine which configuration files **startstone** reads. (For more information, see "How GemStone Uses Configuration Files" on page 350.)

The options for this command generally are not needed for a standard GemStone configuration.

The **-N** option is intended for use when the repository needs recovery, but the transaction logs specified in the configuration file cannot be found or have become corrupted. In such case, **-N** forces the repository to start up anyway, even though transactions committed since the last checkpoint will be lost.

You can use the **-N** option to start a Stone if the transaction files have been lost or corrupted, but the extents are still available. A new transaction log will be initialized as part of the startup.

The **-R** option is intended for use when the repository is restored by starting GemStone on an operating system backup of the extents. The repository monitor is left in a state equivalent to that following restoration of a GemStone backup. You must then invoke Topaz to restore from transaction logs (if available) and commit the restored state. If the extents against which Stone is being started require recovery, or if you are restoring from online extent backups, then you must specify the **-N** option along with the **-R** option; otherwise, an error will result and the repository monitor will not start.

For assistance with startup failures, refer to "To Troubleshoot Stone Startup Failures" on page 129.

statmonitor

statmonitor *stoneName* **-f** *fileName* [*options*]

stoneName	Required; the name of the Stone to monitor.
-A	Collect all available system statistics.
-C	Collect system statistics for each individual CPU.
-D	Collect system statistics for all disks and partitions
-f *fileName*	The output file name. By default, the output filename is statmon*N*.out, where *N* is the process ID. To send output to stdout instead of a file, specify **-f stdout**.
-i *interval*	The interval in seconds (default: 20). Select either **-i** or **-I**.
-I *intervalMs*	The interval in milliseconds (default 20000; minimum 100). Select either **-i** or **-I**.
-h *hours*	The number of hours (default: forever).
-m *stoneHostName*	The Stone host name (default: localHost).
-n *numAppStats*	The number of application statistics (default: 0).
-N	Collect system statistics for all network interfaces.
-p *sessionId*	A GemStone sessionId to monitor. You may specify multiple sessions. (Default: monitor all sessions.)
-P	Sample the Stone, shared cache monitor and all AIO page servers only.
-q	Quiet mode. Only print messages if an error occurs.
-Q*pid1,pid2,...*	Record statistics for a list of process IDs.
-r	Restart a new output file when the current one completes. Each file will be given a unique name. This option may only be used with the **-h** or **-t** switches, which control when a restart is done.
-s0	(deprecated) Same as **-Y**
-s1	(ignored)

-s2	(deprecated) Same as **-D**
-s3	(deprecated) Same as **-D -N**
-s4	(deprecated) Same as **-A**
-S	Sample only the Stone and shared cache monitor.
-t *times*	The maximum number of samples to collect before starting a new output file (default is forever). Select either **-h** or **-t**.
-u *seconds*	The maximum number of seconds to wait before flushing the cached information to the output file (default: 60). If 0 then the flush will be done every interval.
-W	Collect system statistics for system memory pages.
-U*uid1,uid2,...*	Record statistics for all processes owned by one or more UNIX user IDs.
-X	Run in host-only mode. Sample host system statistics only and do not attach to a shared page cache. May be combined with other flags EXCEPT: **-m**, **-n**, **-p**, **-P**, **-S**, or **-Y**.
-Y	Disable collection of all system stats, including per-process data.
-z	Write the output in compressed gzip format.

Record statistics for a repository and/or the operating system to a disk file. **statmonitor** runs in the background, sampling specified data at a specified interval and recording the data to a text file. The data in this file can be viewed by the graphical application VSD.

Statistics are collected from the shared page cache. Only data for GemStone processes attached to the shared page cache on the host on which statmonitor is running are collected. To monitor Gems on systems with remote Gem servers, you must run statmonitor both on the Stone's machine and on the Gem server machine.

The *stonename* argument is used to specify the cache to monitor, both for caches that are local to the Stone and caches that are remote from the Stone with that name. Configurations in which a single machine is hosting remote caches for multiple stones that are running with the same Stone name are ambiguous and will not work correctly; this configuration is strongly discouraged.

For more information on statmonitor and VSD, see Appendix G, beginning on page 457.

stopnetldi

stopnetldi [*netLdiName*] [**-h**]

netLdiName The name of the GemStone network server; the default is `gs64ldi`. Network resource syntax is not permitted.

-h Displays a usage line and exits.

Gracefully stop a GemStone network server. The argument to this command is optional, and are not needed for a standard GemStone configuration.

stopstone

stopstone	[*gemStoneName* [*gemStoneUserName* [*gemStonePassword*]]] [**-i**] [**-t**] [**-h**]
gemStoneName	The name of the GemStone repository monitor, commonly `gs64stone`. Network resource syntax is not permitted.
gemStoneUserName	A privileged GemStone user account name, such as "DataCurator" or "SystemUser".
gemStonePassword	The GemStone password for the specified *gemStoneUserName*.
-i	Causes any current GemStone sessions to be terminated immediately.
-h	Displays a usage line and exits.
-t	Specifies how long to wait for other processes to detach from the cache. Default is -1, wait forever.

Gracefully shut down a GemStone repository monitor. In the process, a checkpoint is performed in which all committed transactions are written to the extents. If you specify the **-i** (immediate) option, the repository monitor is shut down even if there are GemStone users logged in. The -t option specifies how long to wait for all session to detach from the cache before returning; if omitted, **stopstone** will wait forever. If you do not supply the *gemStoneName*, *gemStoneUserName*, and *gemStonePassword* on the command line, this command will prompt you for them.

Because this command uses a Gem session process to connect to *gemStoneName*, it fails if the Gem is unable to connect for any reason, such as inability to attach a shared page cache. For assistance, refer to "To Troubleshoot Session Login Failures" on page 142.

topaz

topaz [-r]	[**-i**] [**-n** *netLdiName*] [**-q**] [**-I** *topazini*] [**-u** *useName*] [**-h**]
topaz -l	[**-i**] [**-n** *netLdiName*] [**-e** *exeConfig*] [**-z** *systemConfig*] [**-T** *tocSizeKB*] [**-q**] [**-I** *topazini*] [**-u** *useName*] [**-h**]
-e *exeConfig*	The GemStone executable-dependent configuration file (applies only to linked sessions). See page 353.
-h	Displays a usage line and exits
-i	Ignore the initialization file, `.topazini`.
-I *topazini*	Specify a complete path and file to a topazini initialization files, and use this rather then any `.topazini` in the default location.
-l	Invoke the linked version of Topaz.
-n *netLdiName*	The name of the GemStone network server; the default is (1) the setting of the `GEMSTONE_NRS_ALL` environment variable; (2) `gs64ldi`
-q	Start Topaz in quiet mode, suppressing printout of the banner and other information.
-r	Invoke the RPC (remote procedure call) version of Topaz.
-T *tocSizeKB*	The GEM_TEMPOBJ_CACHE_SIZE that will be used. Overrides any settings provided in configuration files passed as arguments with the **-e** or **-z** options. Only applies to linked sessions.
-u *useName*	The value is not used by the topaz executable, but may be useful in identifying processes in OS utilities such as `top` or `ps`.
-z *systemConfig*	The GemStone system configuration file (applies only to linked sessions). See page 350.

This command invokes various forms of Topaz. The default is to invoke the remote procedure call (RPC) version of Topaz; to do this explicitly, use the -r option. To invoked the linked version of Topaz, which is recommended or required for some

maintenance operations, use the -l option. Several Topaz arguments only apply in linked Topaz. The arguments **-e***exeConfig* and **-z***systemConfig* determine the configuration files that **topaz -l** reads. For more information about this, see "How GemStone Uses Configuration Files" on page 350.

Settings within topaz can allow linked topaz to perform RPC logins.

For further information, see the *GemStone Topaz Programming Environment*.

waitstone

waitstone [*gemStoneName* | *netLdiName*] [*timeout*]

gemStoneName The name of the GemStone repository monitor, commonly `gs64stone`.

netLdiName The name of the GemStone network server, commonly `gs64ldi`.

timeout How many minutes to wait for GemStone to initialize before reporting a problem. The default (0) means wait forever; -1 means don't wait, try once and return the result. Only valid when specifying either *gemStoneName* or *netLdiName*.

This command reports whether the GemStone repository *gemStoneName* is ready to accept logins or whether *netLdiName* is ready to accept requests. During the first 10 seconds, waitstone tests every two seconds to see if the Stone or NetLDI is ready; thereafter, a new connection is attempted once every 10 seconds. When the service is ready, **waitstone** issues a message to `stdout`. If the service is not ready by the time the specified number of minutes have elapsed, **waitstone** reports an error.

You may specify the *gemStoneName* and *netLdiName* arguments as a GemStone network resource string. (See Appendix C, "Network Resource String Syntax.")

This command returns 0 exit status if the operation is successful; otherwise, it returns a non-zero value.

waitstone does not have a `help` argument, but you can type:

```
man waitstone
```

to display the online manual page.

Appendix

C Network Resource String Syntax

This appendix describes the syntax for network resource strings. A network resource string (NRS) provides a means for uniquely identifying a GemStone file or process by specifying its location on the network, its type, and authorization information. GemStone utilities use network resource strings to request services from a NetLDI.

Overview

One common application of NRS strings is the specification of login parameters for a remote process (RPC) GemStone application. An RPC login typically requires you to specify a GemStone repository monitor and a Gem service on a remote server, using NRS strings that include the remote server's hostname. For example, to log in from Topaz to a Stone process called "gs64stone" running on node "handel", you would specify two NRS strings:

```
topaz> set gemstone !@handel!gs64stone
topaz> set gemnetid !@handel!gemnetobject
```

Many GemStone processes use network resource strings, so the strings show up in places where command arguments are recorded, such as the GemStone log file. Looking at log messages will show you the way an NRS works. For example:

```
Opening transaction log file for read,
filename = !@oboe#dbf!/user1/gemstone/data/tranlog0.dbf
```

An NRS can contain spaces and special characters. On heterogeneous network systems, you need to keep in mind that the various UNIX shells have their own rules for interpreting these characters. If you have a problem getting a command to work with an NRS as part of the command line, check the syntax of the NRS recorded in the log file. It may be that the shell didn't expand the string as you expected.

NOTE

Before you begin using network resource strings, make sure you understand the behavior of the software that will process the command.

See each operating system's documentation for a full discussion of its own rules about escaping certain characters in NRS strings that are entered at a command prompt.

If there is a space in the NRS, you can replace the space with a colon (:), or you can enclose the string in quotes (" "). For example, the following network resource strings are equivalent:

```
% waitstone !@oboe#auth:user@password!gs64stone
% waitstone "!@oboe#auth user@password!gs64stone"
```

Defaults

The following items uniquely identify a network resource:

- Communications protocol— such as TCP/IP
- Destination node—the host that has the resource
- Authentication of the user—such as a system authorization code
- Resource type—such as server, database extent, or task
- Environment—such as a NetLDI, a directory, or the name of a log file
- Resource name—the name of the specific resource being requested.

A network resource string can include some or all of this information. In most cases, you need not fill in all of the fields in a network resource string. The

information required depends upon the nature of the utility being executed and the task to be accomplished. Most GemStone utilities provide some context-sensitive defaults. For example, the Topaz interface prefixes the name of a Stone process with the **#server** resource identifier.

When a utility needs a value for which it does not have a built-in default, it relies on the system-wide defaults described in the syntax productions in "Syntax" on page 424. You can supply your own default values for NRS modifiers by defining an environment variable named GEMSTONE_NRS_ALL in the form of the *nrs-header* production described in the Syntax section. If GEMSTONE_NRS_ALL defines a value for the desired field, that value is used in place of the system default. (There can be no meaningful default value for "resource name.")

A GemStone utility picks up the value of GEMSTONE_NRS_ALL as it is defined when the utility is started. Subsequent changes to the environment variable are not reflected in the behavior of an already-running utility.

When a client utility submits a request to a NetLDI, the utility uses its own defaults and those gleaned from its environment to build the NRS. After the NRS is submitted to it, the NetLDI then applies additional defaults if needed. Values submitted by the client utility take precedence over those provided by the NetLDI.

Notation

Terminal symbols are printed in boldface. They appear in a network resource string as written:

#server

Nonterminal symbols are printed in italics. They are defined in terms of terminal symbols and other nonterminal symbols:

username ::= *nrs-identifier*

Items enclosed in square brackets are optional. When they appear, they can appear only one time:

address-modifier ::= [*protocol*] [**@** *node*]

Items enclosed in curly braces are also optional. When they appear, they can appear more than once:

nrs-header ::= **!** [*address-modifier*] {*keyword-modifier*} **!**

Parentheses and vertical bars denote multiple options. Any single item on the list can be chosen:

protocol ::= (**tcp** | **serial** | **default**)

Syntax

nrs ::= [*nrs-header*] *nrs-body*

where:

nrs-header ::= **!** [*address-modifier*] {*keyword-modifier*} [*resource-modifier*]**!**
All modifiers are optional, and defaults apply if a modifier is omitted. The value of an environment variable can be placed in an NRS by preceding the name of the variable with "$". If the name needs to be followed by alphanumeric text, then it can be bracketed by "{" and "}". If an environment variable named `foo` exists, then either of the following will cause it to be expanded: `$foo` or `${foo}`. Environment variables are only expanded in the *nrs-header*. The *nrs-body* is never parsed.

address-modifier ::= [*protocol*] [**@** *node*]
Specifies where the network resource is.

protocol ::= (**tcp** | **serial** | **default**)
Supports heterogeneous connections by predicating address on a network type. If no protocol is specified, GCI_NET_DEFAULT_PROTOCOL is used. On UNIX hosts, this default is **tcp**.

node ::= *nrs-identifier*
If no node is specified, the current machine's network node name is used. The identifier may also be an Internet-style numeric address. For example:

```
!@120.0.0.4#server!cornerstone
```

nrs-identifier ::= *identifier*
Identifiers are runs of characters; the special characters !, #, $, @, ^ and white space (blank, tab, newline) must be preceded by a "^". Identifiers are words in the UNIX sense.

keyword-modifier ::= (*authorization-modifier* | *environment-modifier*)
Keyword modifiers may be given in any order. If a keyword modifier is specified more than once, the latter replaces the former. If a keyword modifier takes an argument, then the keyword may be separated from the argument by a space or a colon.

authorization-modifier ::= ((**#auth** | **#encrypted**) [**:**] *username* [**@** *password*])

#auth specifies a valid user on the target network. A valid password is needed only if the resource type requires authentication. **#encrypted** is used by GemStone utilities. If no authentication information is specified, the system will try to get it from the `.netrc` file. This type of authorization is the default.

username ::= *nrs-identifier*

If no user name is specified, the default is the current user. (See the earlier discussion of *nrs-identifier*.)

password ::= *nrs-identifier*

If no password is specified, the system will try to obtain it from the user's `.netrc` file. (See the earlier discussion of *nrs-identifier*.)

environment-modifier ::= (**#netldi** | **#dir** | **#log**) [**:**] *nrs-identifier*

#netldi causes the named NetLDI to be used to service the request. If no NetLDI is specified, the default is `gs64ldi`. When you specify the **#netldi** option, the *nrs-identifier* (page 424) is either the name of a NetLDI service or the port number at which a NetLDI is running.

#dir sets the default directory of the network resource. It has no effect if the resource already exists. If a directory is not set, the pattern "`%H`" (home directory) is used. (See the definition of *nrs-identifier* on page 424.)

#log sets the name of the log file of the network resource. It has no effect if the resource already exists. If the log name is a relative path, it is relative to the working directory. If a log name is not set, the pattern "`%N%P%M.log`" (defined below) is used. (See the definition of *nrs-identifier* on page 424.)

The argument to **#dir** or **#log** can contain patterns that are expanded in the context of the created resource. The following patterns are supported:

%H	home directory
%M	machine's network node name
%N	executable's base name
%P	process pid
%U	user name
%%	%

resource-modifier ::= (**#server** | **#spawn** | **#task** | **#dbf** | **#monitor** | **#file**)

Identifies the intended purpose of the string in the *nrs-body*. An NRS can contain only one resource modifier. The default resource modifier is context sensitive. For instance, if the system expects an NRS for a database file, then the default is **#dbf**.

#server directs the NetLDI to search for the network address of a server, such as a Stone or another NetLDI. If successful, it returns the address. The *nrs-body* is a network server name. A successful lookup means only that the service has been defined; it does not indicate whether the service is currently running. A new process will not be started. (Authorization is needed only if the NetLDI is on a remote node and is running in secure mode.)

#task starts a new Gem. The *nrs-body* is a NetLDI service name (such as "gemnetobject"), followed by arguments to the command line. The NetLDI creates the named service by looking first for an entry in `$GEMSTONE/sys/services.dat`, and then in the user's home directory for an executable having that name. The NetLDI returns the network address of the service. (Authorization is needed to create a new process unless the NetLDI is in guest mode.) The **#task** resource modifier is also used internally to create page servers.

#dbf is used to access a database file. The *nrs-body* is the file spec of a GemStone database file. The NetLDI creates a page server on the given node to access the database and returns the network address of the page server. (Authorization is needed unless the NetLDI is in guest mode).

#spawn is used internally to start the garbage collection and other service Gem processes.

#monitor is used internally to start up a shared page cache monitor.

#file means the *nrs-body* is the file spec of a file on the given host (not currently implemented).

nrs-body ::= unformatted text, to end of string

The *nrs-body* is interpreted according to the context established by the *resource-modifier*. No extended identifier expansion is done in the *nrs-body*, and no special escapes are needed.

Appendix

D GemStone Kernel Objects

This appendix describes the predefined objects that are located in a freshly installed GemStone/S 64 Bit repository.

Non-Numeric Constants

The following non-numeric constants are defined in the Globals dictionary and protected by the SystemObjectSecurityPolicy:

- **true** (an instance of Boolean)
- **false** (an instance of Boolean)
- **nil** (an instance of UndefinedObject)

Numeric Constants

Floating point constants are instances of class Float or class DecimalFloat. They are defined in the Globals dictionary and are protected by the SystemObjectSecurityPolicy. Refer to IEEE standards 754-1987 and 854-1987 for more information regarding their meanings in floating-point calculations.

DecimalPlusInfinity
DecimalMinusInfinity
DecimalPlusQuietNaN
DecimalMinusQuietNaN
DecimalPlusSignalingNaN
DecimalMinusSignalingNaN
PlusInfinity
MinusInfinity
PlusQuietNaN
MinusQuietNaN
PlusSignalingNaN
MinusSignalingNaN

Repository and GsObjectSecurityPolicies

SystemRepository. This single instance of Repository is defined in the Globals dictionary. Repository is a subclass of Collection, and the indexable part of SystemRepository contains references to all the security policies (all the instances of GsObjectSecurityPolicy) in GemStone.

The SystemRepository object initially contains eight security policies, three of which are public and named: the SystemObjectSecurityPolicy (owned by the SystemUser), the DataCuratorObjectSecurityPolicy (owned by the DataCurator), and the PublishedObjectSecurityPolicy (owned by SystemUser). The SystemRepository object itself is protected by the DataCuratorObjectSecurityPolicy. New GsObjectSecurityPolicies may be created and added to the SystemRepository object, using the method `newInRepository:`, or by some methods that create new users.

For more on GsObjectSecurityPolicies, see the *Programming Guide for GemStone/S 64 Bit*.

SystemObjectSecurityPolicy. This security policy is defined in the Globals dictionary. For backwards compatibility, the key #SystemSegment also refers to this security policy, and may be used in upgraded repositories. The SystemObjectSecurityPolicy object itself is protected by the DataCuratorObjectSecurityPolicy.

The SystemObjectSecurityPolicy is the default security policy for its owner, the SystemUser (who has write authorization for any of the objects in this security policy). The "world" (that is, the set of all GemStone users) is authorized to read, but not write, the objects in this security policy. In addition, the group #System is authorized to write in this security policy.

DataCuratorObjectSecurityPolicy. This security policy is defined in the Globals dictionary. For backwards compatibility, the key #DataCuratorSegment also refers to this security policy, and may be used in upgraded repositories. The DataCuratorObjectSecurityPolicy object itself is protected by the DataCuratorObjectSecurityPolicy.

The DataCuratorObjectSecurityPolicy is the default security policy for its owner, the DataCurator (who has write authorization for any of the objects in this security policy). The "world" (that is, the set of all GemStone users) is authorized to read, but not write, the objects in this security policy. In addition, the #DataCuratorGroup is are authorized to write in this security policy.

Objects in the DataCuratorObjectSecurityPolicy include the Globals dictionary, the SystemRepository object, most instances of GsObjectSecurityPolicy, AllUsers (the set of all GemStone UserProfiles), AllGroups (the collection of groups authorized to read and write objects in security policies), and each UserProfile object.

PublishedObjectSecurityPolicy. This security policy is defined in the Globals dictionary. For backwards compatibility, the key #PublishedSegment also refers to this security policy, and may be used in upgraded repositories. The PublishedObjectSecurityPolicy object itself is protected by the DataCuratorObjectSecurityPolicy.

The PublishedObjectSecurityPolicy is owned by the SystemUser. The group #Subscribers is authorized to read in this security policy. The group #Publishers is authorized to read and write in this security policy. The "world" is not authorized to read or write the objects in this security policy.

DbfHistory. DbfHistory is a String object that contains information regarding conversions and updates applied to the repository.

NativeLanguage. This Symbol is not used in this version, but may exist in upgraded repositories.

Global Collections

AllGroups. This CanonicalStringDictionary is defined in the Globals dictionary, and is protected by the DataCuratorObjectSecurityPolicy. Each Symbol in AllGroups corresponds to a group of users. When GemStone is first installed, AllGroups contains the symbols #System, #Publishers, #Subscribers, #DataCuratorGroup, and #SymbolUser.

AllUsers. The AllUsers object (a UserProfileSet) is defined in the Globals dictionary, and is protected by the DataCuratorObjectSecurityPolicy. AllUsers contains the UserProfiles of all GemStone users. When GemStone is first installed, AllUsers contains five UserProfiles: SystemUser, DataCurator, GcUser, SymbolUser, and Nameless.

For more information on the Special system accounts, see page 185.

AllDeletedUsers. This collection is defined in the Globals dictionary, and is protected by the DataCuratorObjectSecurityPolicy. AllDeletedUsers contains DeletedUserProfiles representing GemStone users that have been removed from AllUsers. When GemStone is first installed, AllDeletedUsers is empty.

AllClusterBuckets. This ClusterBucketArray is defined in the Globals dictionary, and is protected by the DataCuratorObjectSecurityPolicy. AllClusterBuckets contains instances of ClusterBucket, which group objects on extent pages to improve performance. When GemStone is first installed, AllClusterBuckets contains the following predefined cluster buckets (listed by cluster id):

1. A generic bucket whose extent is “don’t care”. This bucket, the current default after session login, is invariant and may not be modified.
2. A generic bucket whose extent is “don’t care”.
3. A generic bucket whose extent is “don’t care”.
4. The kernel classes “behaviorBucket”, extent 1.
5. The kernel classes “descriptionBucket”, extent 1.
6. The kernel classes “otherBucket”, extent 1.
7. A generic bucket whose extent is “don’t care”.

ConfigurationParameterDict. This SymbolKeyValueDictionary is defined in the Globals dictionary, and is protected by the SystemObjectSecurityPolicy. Its keys list the names of the configuration parameters available to a session. Its values are only used internally in GemStone, to locate the values of the parameters themselves for an individual session.

ErrorSymbols. This SymbolDictionary is defined in the Globals dictionary, and is protected by the SystemObjectSecurityPolicy. It maps mnemonic symbols to error numbers.

GemStoneError. This SymbolDictionary is defined in the Globals dictionary, and is protected by the DataCuratorObjectSecurityPolicy. Each key is a Symbol representing a native language, and is associated with an Array of error

messages in that language. Initially, this dictionary contains the single key #English.

GemStone_Legacy_Streams. This SymbolDictionary contains classes implementing the legacy GemStone PositionableStream interface.

GemStone_Portable_Streams. This SymbolDictionary contains classes implementing the PositionableStream interface that is ANSI-complaint and portable to other Smalltalk dialects.

InstancesDisallowed. This IdentitySet is defined in the Globals dictionary, and is protected by the SystemObjectSecurityPolicy. This collection is used for error reporting for some cases where instance creation is disallowed.

NotTranloggedGlobals. This SymbolDictionary is defined in the Globals dictionary, and is protected by the DataCuratorObjectSecurityPolicy. This collection holds objects for which changes are committed but not recorded in the transaction logs. When GemStone is first installed, NotTranloggedGlobals is empty.

Table D.1 Initial Contents of the Globals Dictionary

	Key	The object's class
Numeric Constants	#DecimalMinusInfinity	DecimalFloat
	#DecimalMinusQuietNaN	DecimalFloat
	#DecimalMinusSignalingNaN	DecimalFloat
	#DecimalPlusInfinity	DecimalFloat
	#DecimalPlusQuietNaN	DecimalFloat
	#DecimalPlusSignalingNaN	DecimalFloat
	#MinusInfinity	Float
	#MinusQuietNaN	Float
	#MinusSignalingNaN	Float
	#PlusInfinity	Float
	#PlusQuietNaN	Float
	#PlusSignalingNaN	Float
Non-Numeric Constants	#false	Boolean
	#nil	UndefinedObject
	#true	Boolean

Table D.1 Initial Contents of the Globals Dictionary (Continued)

	Key	The object's class
Repository and instances of GsObjectSecurityPolicy	#DataCuratorObjectSecurityPolicy, #DataCuratorSegment	GsObjectSecurityPolicy
	#DbfHistory	String
	#PositionableStream_position	String
	#PublishedObjectSecurityPolicy, #PublishedSegment	GsObjectSecurityPolicy
	#SystemRepository	Repository
	#SystemObjectSecurityPolicy, #SystemSegment	GsObjectSecurityPolicy
Collections	#AllClusterBuckets	ClusterBucketArray
	#AllDeletedUsers	IdentitySet
	#AllGroups	CanonicalStringDictionary
	#AllUsers	UserProfileSet
	#ConfigurationParameterDict	SymbolKeyValueDictionary
	#ErrorSymbols	SymbolDictionary
	#GemStoneError	SymbolDictionary
	#GemStone_Portable_Streams	SymbolDictionary
	#GemStone_Legacy_Streams	SymbolDictionary
	#Globals	SymbolDictionary
	#InstancesDisallowed	IdentitySet
	#LegacyErrNumMap	Array
	#NotTranloggedGlobals	SymbolDictionary

Table D.1 Initial Contents of the Globals Dictionary (Continued)

	Key	The object's class
GemStone Internal Objects	#AsciiCollatingTable	ByteArray
	#ConversionReservedOopMap	Array
	#ConversionStatus	Array
	#DoubleByteAsciiCollatingTable	DoubleByteString
	#FdcResults	UndefinedObject
	#GcCandidates	UndefinedObject
	#GcCandidatesCount	UndefinedObject
	#GcHints	UndefinedObject
	#GcWeakReferences	Array
	#GemStoneRCLock	Object
	#GsCompilerClasses	SymbolDictionary
	#GsIndexingObjectSecurityPolicy, #GsIndexingSegment	GsObjectSecurityPolicy
	#ImageVersion	SymbolDictionary
	#isInternedNeedsSymbolCheck	Boolean
	#ObsoleteClasses	SymbolDictionary
	#QuadByteAsciiCollatingTable	QuadByteString
	#RcBtreeNode	UndefinedObject
	#_remoteNil	UndefinedObject
	#SecurityDataObjectSecurityPolicy, #SecurityDataSegment	GsObjectSecurityPolicy
	#SharedDependencyLists	DepListTable
	#VersionParameterDict	SymbolKeyValueDictionary
plus all kernel classes		

Current TimeZone

Each instance of DateTime includes a reference to a TimeZone object, which handles the conversion from the internally stored Greenwich Mean Time (GMT, also referred to as UTC or Coordinated Universal Time) and the local time. TimeZones encapsulate the daylight savings time (DST) rules, so a given GMT time is adjusted to local time based on TimeZone and the specific date. TimeZones are also used to calculate the internal stored GMT for newly created DateTime instances.

Each session has a current TimeZone and a default TimeZone, which are used to display times, and in DateTime creation when methods that do not explicitly specify the TimeZone are used. These are installed as part of application installation or configuration; by default, the GemStone distribution has the America/Los_Angeles TimeZone installed. This is described in the *GemStone/S 64 Bit Installation Guide*.

GemStone uses the public domain **zoneinfo** database to create TimeZone, loading the information from platform and language independent source files. If the rules change for the TimeZone that your application uses, you must recreate the TimeZone instance from the source files. Depending on the nature of the rules change, you may also need to update references from DateTime instances to the new TimeZone instance, or possibly update the DateTime internal offsets.

There are a number of ways to create TimeZone instances for your application:

- **From the OS on Solaris or Linux**. On these operating systems, you can create the TimeZone instance based on the current machine configuration using:

  ```
  newTZ := TimeZone fromOS
  ```

- **GemStone's time zone database.** Using the interactive script `tzselect`, you can determine the correct time zone descriptor name for your local time zone. With this, you can create the new TimeZone instance using the time zone database provided with GemStone.

  ```
  newTZ := TimeZone fromGemPath: '$GEMSTONE/pub/timezone/etc/
  zoneinfo/Europe/Zurich'
  ```

- **Your own time zone database**. With the time zone descriptor name for your TimeZone, you can specify the full path to the time zone information.

  ```
  newTZ := TimeZone fromGemPath: yourPath, '/Europe/Zurich'.
  ```

You must then install this TimeZone instances as the current and default time zone.

Zoneinfo

The widely used public-domain time zone database, **ZoneInfo** or **tz**, contains code and data that records time zone information for locations worldwide. It is updated periodically when boundaries or rules change in any of the represented locations.

Each record in the tz database represents a location where all clocks are kept on the same time as each other throughout the year, coordinating any time adjustments such as DST, and have done so for many years. Locations are

identified by continent (or ocean, for islands) and name, which is usually the largest city within the region. For example, America/Los_Angeles, Europe/London, etc.

tz is provided as text files, which may be compiled into binary files using tz's compilers. GemStone's TimeZone implementation uses the compiled binary form, which is also used by the Solaris and Linux operating systems. To get updated source files, download from

```
ftp://elsie.nci.nih.gov/pub/tzdata*.tar.gz
```

Shipped files are based on `tzdata2006p.tar.gz`. The timezone sources in this file may be compiled using the zic timezone compiler, which GemStone provides as a convenience (see "zic" on page 436).

Utilities

tzselect, **zdump** and **zic** are public domain, open source utilities that are useful in working with the zoneinfo database. These utilities are provided with the Solaris and Linux operating systems; for the convenience of users on other operating systems, these utilities are provided along with the other zoneinfo database files.

NOTE

These are not GemStone utilities. Support for their use is not provided by GemStone.

You may download the source code for these utilities here:

```
ftp://elsie.nci.nih.gov/pub/tzcode*.tar.gz
```

Shipped files are based on `tzcode2006p.tar.gz`. Documentation for these utilities is provided as man pages. To read the man pages, add the directory $GEMSTONE/pub/timezone/man to the MANPATH.

To run these, you may wish to add $GEMSTONE/pub/timezone/etc to the executable path.

tzselect

tzselect allows to you interactively select a time zone. The interactive script asks you a series of questions about the current location and outputs the resulting time zone description to standard output. The output is suitable as a value for the TZ environment variable and GemStone scripts.

You may need to set the environment variable $TZDIR to $GEMSTONE/pub/timezone/etc/zoneinfo (or the path to your zoneinfo

database, for this script to work correctly. You may also need to set the environment variable $AWK, to any POSIX compliant awk program.

For further details on using tzselect see the man page.

zdump

```
zdump [-v] [-c cutoffyear] [zonename...]
```

zdump prints time zone information. It prints the current time for each time zone (zonename) listed on the command line.

Specifying an invalid zone name to zdump does NOT return an error; instead, it returns the zdump output for GMT. This reflects the same behavior of the time routines in libc.

The -v option will display the entire contents of the time zone database for the given time zone name.

For further details on using zdump, including the command line options, see the man page.

zic

```
zic [-s] [-v] [-l localtime] [-p posixrules]  [-d directory]
[-y yearistype] [filename...]
```

zic compiles time zone source files. It reads input text in files named on the command line, and creates the time zone binary files.

To create files in a specific location, rather than the standard platform directory (on Solaris, /usr/share/lib/zoneinfo), use the -d *directory* option.

For example, to recompile sources on Solaris to the GemStone timezone database, execute the following:

```
zic -d $GEMSTONE/pub/timezone/etc/zoneinfo/
/usr/share/lib/zoneinfo/src/northamerica
```

For further details on using zic, including the command line options and the structure of the source code files, see the man page for zic.

Appendix

E Environment Variables

This appendix lists the environment variables used by GemStone/S 64 Bit. The list has two parts: variables intended for public use, and variables that are reserved for internal use.

Public Environment Variables

The following environment variables are intended for use by customers. The variable GEMSTONE is required; the others may be useful in particular situations.

GEMSTONE
: The location of the GemStone Object Server software, which must be a full path, beginning with a slash, such as `/user3/GemStone-hppa.hpux`.

GEMSTONE_ADMIN_GC_LOG_DIR
: The directory location for Admin GcGem logs. By default, Admin GcGem log files are created in the same directory as the Stone log, which is $GEMSTONE/data. You can set this environment variable to specify that Admin GcGem log files are created in a different directory.

GEMSTONE_CHILD_LOG
: Used by parent process to tell child process the name of its log file. To keep the child's log from being deleted when the process terminates normally, unset

this variable in the appropriate script, such as `$GEMSTONE/sys/gemnetobject`.

GS_DISABLE_SIGNAL_HANDLERS
: When this environment variable is set, gems that encounter a SIGSEGV, SIGBUS or SIGILL signal do not attempt to handle it. The session will immediately exit, without printing stacks to its log and without generating a core file.

GEMSTONE_EXE_CONF
: The location of an executable-dependent configuration file; see "Creating an Executable Configuration File" on page 354.

GEMSTONE_GLOBAL_DIR
: The location for the global GemStone logs and locks file, overriding the default /opt/gemstone/. This directory must already exist.

  This directory controls visibility between GemStone processes, and must be the same for all GemStone processes that may want to interact with a given repository, including stone, gems, topaz, statmonitor, netldi, gslist, etc. GemStone processes that do not shared a common location for /gemstone/locks—either /opt/gemstone, /usr/gemstone, or a directory specified by $GEMSTONE_GLOBAL_DIR—operate as if they are in independent name spaces.

GEMSTONE_LIB
: Specifies the directory for the gem and gem dynamic libraries. This is primarily of use in debugging low-level problems, and is used by the gemnetdebug script to specify the slow gem and gem dynamic libraries.

GEMSTONE_LOG
: The location of system log files for the Stone repository monitor and its child processes. For further information, see "GemStone System Logs" on page 156.

GEMSTONE_MAX_FD
: Limits the number of file descriptors requested by a GemStone process. For further information, see "Estimating File Descriptor Needs" on page 36.

GEMSTONE_NRS_ALL
: Sets a number of network-related defaults, including the type of user authentication that GemStone expects. For further information, see "To Set a Default NRS" on page 105.

GEMSTONE_PAGE_MGR_LOG_DIR
: The directory location for Page Manager Gem logs. By default, Page Manager Gem log files are created in the same directory as the Stone log, which is

$GEMSTONE/data. You can set this environment variable to specify that Page Manager Gem log files are created in a different directory.

GEMSTONE_RECLAIM_GC_LOG_DIR
The directory location for all Reclaim GcGem logs. By default, Reclaim GcGem log files are created in the same directory as the Stone log, which is $GEMSTONE/data. You can set this environment variable to specify that Reclaim GcGem log files are created in a different directory.

GEMSTONE_SPCMON_STARTUP_TIMELIMIT
Internally, GemStone waits for five minutes for the shared page cache to start up and initialize. This environment variable overrides this timeout, and specifies the time, in seconds, that the Stone will wait for the shared page cache to startup before giving up.

GEMSTONE_SYMBOL_GEM_LOG_DIR
The directory location for SymbolGem logs. By default, Symbol Gem log files are created in the same directory as the Stone log, which is $GEMSTONE/data. You can set this environment variable to specify that Symbol Gem log files are created in a different directory.

GEMSTONE_SYS_CONF
Location of a system-wide configuration file; see "How GemStone Uses Configuration Files" on page 350.

GS_CORE_TIME_OUT
If GS_WRITE_CORE_FILE is defined, this is the number of seconds to wait before a catastrophically failing GemStone/S process writes a core file and terminates—by default, 60 seconds. To determine the cause of a problem, GemStone/S Technical Support needs a stack trace, which is usually written to the process log file prior to the process shutdown.

If you need to derive a stack trace directly from a failing (but not yet terminated) process by attaching a debugger to it, you can set this variable to increase the time available to attach the debugger.

GS_DEBUG_VMGC_MKSW_MEMORY_USED_SOFT_BREAK
At the end of each mark/sweep, if the percent of memory used is greater than the threshold specified by this variable, a SoftBreak (error 6003) is generated, and the threshold is raised by 5 percent. We suggest a setting of 75%.

GS_DEBUG_VMGC_MKSW_PRINT_STACK
The mark/sweep count at which to begin printing the Smalltalk stack at each mark/sweep.

For this and all other GS_DEBUG_VMGC_* environment variables, the printout goes to the "output push" file of a linkable Topaz (topaz -l) session, for use in testing your application. If that file is not defined, the printouts go to standard output of the session's gem or topaz -l process.

GS_DEBUG_VMGC_MKSW_PRINT_C_STACK
The mark/sweep count at which to begin printing the C stack at each mark/sweep. This variable is very expensive, consuming 2 seconds plus the cost of fork() for each printout.

GS_DEBUG_VMGC_PRINT_MKSW
The mark/sweep count at which to begin printing mark/sweeps.

GS_DEBUG_VMGC_PRINT_MKSW_MEMORY
The mark/sweep count at which to begin printing detailed memory usage (20 lines) for each mark/sweep.

GS_DEBUG_VMGC_PRINT_MKSW_MEMORY_USED
Specifies when Smalltalk stack printing starts as the application approaches OutOfMemory conditions. At the end of each mark/sweep, if the percent of memory used is greater than the threshold specified by this variable, the mark/sweep is printed, the Smalltalk stack is printed, and the threshold is raised by 5 percent. In a situation producing an OutOfMemory error, you should get several Smalltalk stacks printed in the Gem log file before the session dies. The default setting is 75%.

GS_DEBUG_VMGC_PRINT_SCAV
The scavenge count at which to begin printing scavenges. Once this takes effect, all mark/sweeps will also be printed. Be aware that printing scavenges can produce large quantities of output.

GS_DEBUG_VMGC_PRINT_TRANS
Print transaction boundaries (begin/commit/abort) in the log file.

GS_DEBUG_VMGC_SCAV_PRINT_STACK
The scavenge count at which to begin printing the Smalltalk stack at each scavenge. Be aware that this print activity can produce large quantities of output.

GS_DEBUG_VMGC_SCAV_PRINT_C_STACK
The scavenge count at which to begin printing the C stack at each scavenge. This variable is very expensive, consuming 2 seconds plus the cost of fork() for each printout.

GS_DEBUG_VMGC_VERBOSE_OUTOFMEM
Automatically call the primitive for

`System class>>_vmPrintInstanceCounts:0` when an OutOfMemory error occurs, and also print the Smalltalk stack. (For details about this method, see the comments in the image.) This applies to each Gem or linkable Topaz (topaz -l) process that you subsequently start.

GS_DEBUG_VMGC_VERIFY_MKSW
: The mark/sweep count at which to begin verifying object memory before and after each mark/sweep.

GS_DEBUG_VMGC_VERIFY_SCAV
: The scavenge count at which to begin verifying object memory before and after each scavenge. Once this takes effect, GS_DEBUG_VMGC_VERIFY_MKSW will also be in effect. Be aware that this activity uses significant amounts of CPU time.

GS_DISABLE_CHARACTER_TABLE_LOAD
: Any value disables the loading of any customized Extended Character Set Character Data Tables. See "Disabling load of the Character Data Table" on page 454.

GS_DISABLE_KEEPALIVE
: A non-empty string disables the network keepalive facility. For further information about keepalive, see "Disrupted Communications" on page 98.

GS_DISABLE_WARNING
: A non-empty string disables a warning that GemStone is using `/opt/gemstone` instead of `/usr/gemstone` for log and lock files when both directories exist.Use of `/usr/gemstone` is only for compatibility with previous releases; the default location is `/opt/gemstone`.

GS_PAGE_MGR_PRINT_REMOTE_STACKS
: If this variable is set, if a remote cache page server becomes stuck, the page manager will request that the remote cache page server print its call stack to its log file.

GS_WRITE_CORE_FILE
: By default, core files are not created when a fatal error occurs. (The C level stack trace is written to the process log file prior to the process shutdown.) You can set this environment variable if you need a core file.

upgradeLogDir
: The location for log files produced during the upgrade of a repository for a new version of GemStone.

System Variables Used by GemStone

GemStone uses the following system variables that exist for other purposes:

EDITOR	Used by Topaz to determine which editor to invoke.
PATH	The search path of locating executable files.
SHELL	Used to determine what shell to use for an exec, such as by `System class>>performOnServer:`.

Reserved Environment Variables

The following environment variables are reserved for internal use. Customers should not define these variables for use with GemStone unless specifically instructed to do so. Please refrain from using these variables for other purposes.

_POSIX_OPTION_ORDER

GCIRTL_BASELIBNAME

GEMSTONE*
: All environment variable names beginning with “GEMSTONE” other than those above are reserved.

GS_*
: All environment variable names beginning with “GS_” other than those above are reserved.

NT_PARENT_PID

netldi*nn*

runpgsvr

Appendix

F Localization

This appendix addresses the following topics related to localization:

- GemStone Smalltalk class Locale (page 443)

  How to obtain and override operating system locale information in GemStone.

- Extended Character Set support (page 445)

  How to process and collate String data that includes extended Characters (that is, beyond the basic set of 256 Characters).

F.1 Class Locale

The class Locale allows you to obtain operating system locale information and use or override it in GemStone. GemStone currently only uses the decimalPoint setting, to provide localized reading and writing of numbers involving decimal points. Note that Smalltalk syntax requires the use of '.' as the decimal point separator, so expressions involving literal floating point numbers within Smalltalk code will still require use of the '.', regardless of Locale.

To override the operating system locale information, use the following message:

```
Locale class >> setCategory: categorySymbol locale: LocaleString
```

The *LocaleString* passed to `setCategory:locale:` must be defined on the host machine. You can use the UNIX command **locale -a** to get a list of all available LocaleStrings.

Table F.1 lists the valid category symbols.For more information about these settings, see the operating system man page for locale.

Table F.1 Valid Category Symbols for Locale

#LC_CTYPE	Character handling
#LC_NUMERIC	Decimal point handling
#LC_TIME	Date and time handling
#LC_COLLATE	Collation setting
#LC_MONETARY	Monetary handling
#LC_MESSAGES	Messages handling
#LC_ALL	All locale categories

To use decimal localization appropriate for Germany:

```
Locale setCategory: #LC_NUMERIC locale: 'de_DE'.
```

To use UNIX default values:

```
Locale setCategory: #LC_ALL locale: 'C'.
```

The following method returns a Locale object with the locale information for the current session:

```
Locale class >> current
```

The following method returns the decimalPoint setting for the current Locale:

```
Locale decimalPoint
```

NOTE

You can use similar methods to obtain settings for other fields, but only decimalPoint is significant in this release.

By passing in an argument of false, the following methods allow you to convert a String to a Float without consideration of the current Locale setting:

```
Float class >> fromString:useLocale:

DecimalFloat class >> fromString:useLocale:
```

For more information, see the image and operating system documentation for locale (man `localeconv`).

F.2 Extended Character Set Support

GemStone/S 64 Bit supports the ability to process and collate String data that includes extended Characters (that is, beyond the basic set of 256 Characters). This allows the various Character test, comparison, and conversion methods to work correctly across the full range of Characters that can be represented in up to four bytes. You can control the contents of the character data tables that drive these methods, and can either use tables based on the Unicode Standard or design your own, based on your particular application requirements.

If you do not require more than the basic 256 Character set, you do not need to do anything. By default, GemStone uses basic 256 Character set tables, which provide the best performance. Support for Extended Character Set provides additional abilities, without compromising performance for applications that do not need them.

Extended Character Set support includes:

- Default Built-in Tables — By default, GemStone/S 64 Bit comes with basic character set tables for the 0–255 Character range, based on the Unicode Standard.
- 32-bit Unicode Standard — You can extend the character data tables to support the Unicode Standard including all Characters that can be represented with four bytes (0–2147483647).
- Flexible upgrade and customization — You can build your own character data tables, or update your applications when there are new releases of the Unicode Standard.

Extended Character Sets

GemStone Characters are based on an index, which correspond to the Unicode value. By themselves, Characters can only be compared by this index, which does not provide a good way to determine which should be sorted first, and it does not provide any information about uppercase or lowercase, or if the Character is a digit or letter.

To supply this information, GemStone provides a built-in internal table that includes the collation information for the standard 256 Characters, in a legacy collate order. This internal table provides the best possible performance for sorting or indexing of String data.

Characters that are not in the currently loaded table can still be compared and sorted, but the ordering is based on the Character value. So Character such as š that

are outside the standard 256 Character range are not collated correctly by the default table. In addition, there are Characters within the standard 256 Character table that are not collated correctly for some languages; for example, ß (Character withValue: 223) by default is collated at the end of the alphabet.

If your application uses data that contains Characters such as ß and š, or any Characters beyond the standard 256 Characters, you can easily load Character Data Tables that provide collation, conversion, and querying, up to the limit of the Character set you will be using.

Character Data Tables

To hold and order the Character relationship information that is needed for collation, GemStone defines formatted Array structures referred to as Character Data Tables.

Internally, GemStone uses an primitive Character Data Table, composed of an Array of ByteArrays that contain mapping and numeric references. This format uses the minimum memory and permits the fastest access.

To allow the tables to be examined and customized, a Character Data Table can be converted into a structured form, consisting of an Array of Arrays, where each element Array contains the information for a Character.

When customized Character Data Tables are installed, either from a file or from a structured Character Data Table, the Globals Symbol Dictionary variable #CharacterDataTables is set to the primitive Character DataTable. If the variable #CharacterDataTables is nil, the default built-in 256 Character table is used. This variable is checked by each session during login, and remains in use for the lifetime of the session.

Loading Extended Character Sets

The GemStone distribution includes a file with a Character Data Table containing the complete Unicode Character Data Set, which you can load into your repository to allow you to immediately work with all Unicode Characters.

GemStone indexes and SortedCollections depend on a static collation sequence. Use caution when loading new tables. Before making changes in the Character Data Tables, all indexes on the system should be dropped, and after the new tables are installed, the indexes can be rebuilt, and SortedCollections updated.

The Character Data Tables can only be updated by SystemUser, and the changes are global for all users on a system.

To load the complete Unicode Data Set into your image:

1. Log in as SystemUser
2. execute the following:

```
Character activateCharTablesFromFile:
   '$GEMSTONE/examples/CharTableUnicode510.dat'
```

3. Commit and log out. The new tables are in effect for subsequent logins.

While this is an easy way to load all the Character information that is available, the complete table includes over 100K Characters, which is likely to be more than your application actually needs. This may consume an unnecessary amount of memory.

We recommend installing customized tables containing a subset of the full Unicode Data. However, modifying the Character Data Tables should be done rarely, as it requires index and SortedCollection rebuild. All Characters that may potentially be needed in the future by your application should be included in the customized tables.

Customizing the Character Data Table

By default, GemStone Character Data Tables are in the legacy collation sequence, to avoid problems with upgraded applications. The updated Unicode Character Data Table file, `CharTableUnicode510.dat`, includes the Unicode default collation table, DUCET. This provides a much improved default collation over the legacy collation.

However, the DUCET collation will not be correct for all national languages under all uses. GemStone allows you go customize the collation sequence if necessary.

There are several ways to customize the collation sequence.

From within GemStone, you can create a Structured Character Data Table based on the default table, or on the Unicode primitive data tables, and edit this using GemStone Smalltalk expressions. When you have modified the table to meet your needs, you can install the table into your system.

Alternatively, you can download the Unicode data and default collation table text files. After editing this raw data in text form outside of GemStone, you can use GemStone utility code to generate your own Character table .dat files, and install these files into your system.

Structured Character DataTable

The structured Character data tables allow a System Administrator to view and update the collation sequence in an easy to understand format.

A Structured Character Table is composed of an Array of elements, arranged according to character collation order. Each element is an Array containing the following entries:

- The character for this entry.
- The symbolic character category code.
- Uppercase character (if a letter) / Numerator (if numeric).
- Lowercase character (if a letter) / Denominator (if fraction).
- (Optional) Titlecase character

For example, in the default Structured Character Table, the 73rd element is the letter $a (Unicode/ASCII 97). The array at this index contains the following:

```
#( $a #Ll $A $a )
```

The first element is the Character itself; the second is the category, which in this example is lowercase, followed by the uppercase and lowercase equivalents.

To create a structured character data table based on the currently installed primitive Character Data Table, execute:

```
Character charTables
```

Once the Structured Character Data Table is correct, you can install it into your system by executing:

```
Character installCharTables: aStructudedCharacterDataTable
```

Example

Say we want to add the Characters Š (Unicode 352) and š (Unicode 353) to the standard 256-Character library. These Characters should follow S and s in collation sequence, respectively.

1. Create an instance of Structured Character Table, and place this in a temporary variable. You can generate structured Character Data Tables based on the tables currently in use using this code:

   ```
   UserGlobals at: #MyCharacterDataTables put: Character
   charTables
   ```

   In this case, since there is no customized Character Data Table already loaded, `Character charTables` returns a structured table based on the default built-in 256 Character table.

2. Add the required entries to this table. Each entry is an Array formatted as described on page 447. The entries must be positioned in the Table (Array of Arrays) in the desired sort order, not added to the end (unless the new Character should be sorted last in any collation sequence).

```
MyCharacterDataTables
   add: (Array
         with: (Character withValue: 352)
         with: #Lu
         with: (Character withValue: 352)
         with: (Character withValue: 353))
   after: #( $S #Lu $S $s ).
MyCharacterDataTables
   add: (Array
         with: (Character withValue: 353)
         with: #Ll
         with: (Character withValue: 352)
         with: (Character withValue: 353))
   after: #( $s #Ll $S $s ).
```

3. When you are sure your changes are correct, as SystemUser, execute the following and commit:

```
Character installCharTables: MyCharacterDataTables
```

This converts the Structured Character Table into equivalent byte arrays and places them into the Globals variable #CharacterDataTables, to be loaded by all later sessions on login.

CAUTION

It is possible to affect or break indexes that rely on String sequencing. For this reason, we recommend you that drop any indexes based on Strings in your System before modifying the character tables.

Generating Tables from Unicode Data

To generate tables from the Unicode text files, you must load utility code that GemStone provides as an optional extension. This code is used to generate structured and primitive Character Data Tables based on the Unicode-standard raw text files.

This technique is used both for creating customized versions of the Unicode tables, as well as to rebuild the tables when a new version of the Unicode standard is released.

1. Download the Unicode data files:

   `http://www.unicode.org/Public/UNIDATA/UnicodeData.txt`

   and save to a local file, such as `myUnicodeData.txt`.

   Download the Unicode collation file:

   `http://www.unicode.org/Public/UCA/latest/allkeys.txt`

   and again, save to a local file, such as `myUnicodeCollation.txt`.

2. Optionally, edit `myUnicodeData.txt`, or `myUnicodeCollation.txt`, to modify the collation, reduce the size, or make any other changes necessary for your application.

3. Start topaz, and login as SystemUser. Load the code to build Character Data Tables

   ```
   topaz 1> input $GEMSTONE/examples/UnicodeData.gs
   ```

   and commit

4. Determine how many Characters your application needs to support and should be loaded into the tables. In this example, we will use 65535, which includes all DoubleByteString Characters, but not QuadByteString Characters. If you Strings in your application require fewer Characters, you may be able to make this number much smaller.

5. Execute the following. This will read the data files, generate the tables, and install the tables into your system.

   ```
   | tables|
   UnicodeData
     loadFromFile: 'myUnicodeData.txt'
     max: 65535
     sort: 'myUnicodeCollation.txt'.
   tables := UnicodeData generateTables.
   Character installCharTables: tables.
   ```

6. Commit, and log out. The changes will take effect on the next login, for all users on this system.

The Unicode Database

The Unicode Consortium is an international standards organization that produces the Unicode Database, which provides unique codes for all Characters in all Character Sets. The database continues to develop; GemStone/S 64 Bit 3.0 uses Unicode version 5.1.

For more information on this database, refer to:

```
http://www.unicode.org/Public/UNIDATA/UCD.html
```

The Unicode Consortium provides code charts by script as well as a single master list of all characters, presented in an ASCII-only, comma-delimited version. Version 5.1 of this file has been used to create the ByteArray tables, and is provided in passivated object form.

This table can be found at

```
http://www.unicode.org/Public/UNIDATA/UnicodeData.txt
```

Character Categories

In addition to assigning unique codes for all Characters, and providing the upper and lowercase mappings, the Unicode Database also provides categories for all possible Characters. These categories allow letters to be distinguished from numeric characters and from punctuation.

Table F.2 lists all the Unicode categories. While some of these Character categories are commonly used, other categories are more rare and are not used in English or Latin-based languages.

Table F.2 Character Category Codes and Symbols

1	#Lu	Letter, Uppercase
2	#Ll	Letter, Lowercase
3	#Lt	Letter, Titlecase
4	#Lm	Letter, Modifier
5	#Lo	Letter, Other
6	#Mn	Mark, Nonspacing
7	#Mc	Mark, Spacing Combining
8	#Me	Mark, Enclosing
9	#Nd	Number, Decimal Digit
10	#Nl	Number, Letter

Table F.2 Character Category Codes and Symbols (Continued)

11	#No	Number, Other
12	#Pc	Punctuation, Connector
13	#Pd	Punctuation, Dash
14	#Ps	Punctuation, Open/Start
15	#Pe	Punctuation, Close/End
16	#Pi	Punctuation, Initial Quote
17	#Pf	Punctuation, Final Quote
18	#Po	Punctuation, Other
19	#Sm	Symbol, Math
20	#Sc	Symbol, Currency
21	#Sk	Symbol, Modifier
22	#So	Symbol, Other
23	#Zs	Separator, Space
24	#Zl	Separator, Line
25	#Zp	Separator, Paragraph
26	#Cc	Other, Control
27	#Cf	Other, Format
28	#Cs	Other, Surrogate
29	#Co	Other, Private Use
30	#Cn	Other, Not Assigned

Categories are used to answer these queries:

```
Character >> isDigit
Character >> isLetter
Character >> isLowercase
Character >> isUppercase
Character >> isSeparator
```

Titlecase category

In addition to uppercase and lowercase, the Unicode Standard provides the case-related category Titlecase (#Lt). This is a special case for composite characters that incorporate multiple individual characters—for example, the Unicode Character code 0x01C5 (decimal 453), DZ. (Such characters are rare; there are only 12 such

characters in the Unicode Standard.) Titlecase of these composite character forms consists of an Uppercase first character, followed by all remaining characters in lowercase. In this example, the uppercase would be DŽ, the lowercase would be dž, and the titlecase Dž.

For most letters, Titlecase is the same as Uppercase. Titlecase is an optional feature in GemStone's formatted tables.

GemStone Character Table Data Files

The $GEMSTONE/goodies directory contains two files in GemStone passivate format:

`CharTableDefault.tab`
: Character table data for the default table, in structured form, in legacy collate order.

`CharTableUnicode410.dat`
: Character table data for 16-bit Unicode, in low-level ByteArray form, in legacy collate order. You can use this file to install the extended Unicode tables. (See "Loading Extended Character Sets" on page 446.)

`CharTableUnicode510.dat`
: Character table data for 16-bit Unicode, in low-level ByteArray form, in DUCET collate order. (See "Loading Extended Character Sets" on page 446.)

`UnicodeData.gs`
: This includes the GemStone Smalltalk source code to generate Character Data Tables based on the raw Unicode text files.

Archiving and distributing Character Data Tables

Character Data Tables can be stored in operating system files in passive object format, and loaded from these passivated files. This allows you to archive and distribute the specific Character sets and collation your application is using.

To save the current contents of the Globals variable #CharacterDataTables to a passivated file, execute:

```
Character passivateCharTablesToFile: 'MyFileName'
```

To install a previously passivated from a file, log in as System user and execute:

```
Character activateCharTablesFromFile: 'MyFileName'
```

Commit and log out.

Troubleshooting

Restoring the default table

If you want to clear out the Character DataTables, and return to using the default built-in 256 Character tables, do the following:

1. Log in as SystemUser.
2. Execute:

```
Globals removeKey: #CharacterDataTables.
```

Commit, and log out.

Disabling load of the Character Data Table

When Character Data Tables are loaded during login, GemStone can sometimes detect if an invalid Character Data Table is installed and will revert back to the default 256 character data table. But in other cases, invalid tables may cause problems interpreting Smalltalk source code or topaz commands. In these cases you may need to manually disable the loading of the character data table.

To recover from serious problems after installing an invalid character data table, do the following:

1. At the operating system level, set the host environment variable GS_DISABLE_CHARACTER_TABLE_LOAD to TRUE. The particular value of this environment variable does not matter; its existence is the critical factor.
2. Start a new Topaz session and log in as SystemUser.
3. Execute the following:

```
Globals at: #CharacterDataTables put: nil.
```

4. Commit.

Use caution when using this environment variable. Like any use of a different collation sequence, data structures that depend on collation, such as indexes, will not work correctly and can become inconsistent if modified.

Writing Extended Characters to Files

Characters outside the base ASCII range of 0...255 require multiple bytes to represent. Strings containing Characters with values over 255 are transparently morphed into DoubleByteStrings or QuadByteStrings, depending on the number of bytes required to hold the largest Character in the String.

GsFile allows Strings to be written to external files. GsFile writes bytes, and does not automatically handle DoubleByteStrings or QuadByteStrings. To write data containing Characters outside the base ASCII range to external files, in such a way that the text can be read in again later, they must be explicitly encoded into UTF-8 before writing to the file.

To encode a String, DoubleByteString, or QuadByteString into UTF-8, use `encodeAsUTF8`. To decode a String that is in UTF-8 encoding, send `decodeFromUTF8`.

Appendix

G statmonitor and VSD Reference

This appendix provides details about VSD and statmonitor:

- How to use the statmonitor and Visual Statistics Display (VSD) utilities to evaluate GemStone performance (page 457)
- VSD menu reference (page 471)
- Descriptions of VSD files, including the syntax for template filters (page 473)

G.1 Using statmonitor and VSD

Every commercial aircraft in service today contains a flight data recorder, commonly called a black box. Black boxes are used to help determine what went wrong if the aircraft ever crashes. Without it, crash investigators would have very little to go on and the cause of many crashes would never be known.

In a production GemStone application, the *statmonitor* program serves the role of the black box. It runs in the background, logging the values of all cache statistics to a text file. Should a problem with the repository occur, you can review these statistics to determine the cause, allowing you to identify problems and tune GemStone applications for better performance.

Many questions about repository performance are impossible to answer after the fact unless statmonitor log files are available. For example:

- Why did `markForCollection` take longer than normal?
- Why did the backup run slower than before?
- Why was there more repository growth than normal?
- Which Gem session was consuming the most CPU or I/O during a time of poor performance?
- Which Gem had the most number of commit failures?

If you're trying to answer questions such as these, statmonitor statistics files are your best source of information. statmonitor reads the statistics from the shared page cache that are recorded by GemStone processes, and writes those statistics to a file.

To help you interpret the statistical information that statmonitor gathers, GemStone provides a tool called *VSD* (Visual Statistical Display). VSD's graphical user interface gives you a meaningful way to see how your GemStone system changes over time, based on periodic samples of the system's state. VSD reads the statmonitor files and displays the values in a graph. VSD can also start statmonitor, if it is not already running, and read the values as soon as statmonitor collects them.

statmonitor and VSD are companion tools:

- *statmonitor* is a stand-alone executable with a command-line interface. It does not log into GemStone nor use a GemStone session. However, it is version-sensitive; you must use the version of statmonitor appropriate for your GemStone version.
- VSD provides a graphical user interface for viewing the text files that statmonitor produces. It is independent of the operating system platform, version and server product that created the statmonitor flat file. However, older versions of VSD cannot always read statmonitor files produced by more recent GemStone server versions.

  On UNIX platforms, VSD requires X-Windows. Although the GemStone/S 64 Bit server does not support Windows, you can view statmonitor files generated by GemStone/S 64 Bit using VSD on Windows.

statmonitor uses a discrete sampling algorithm. Events that occur between samples are not recorded.

Starting statmonitor

To start statmonitor, at the command line, enter:

`statmonitor` *stoneName* `-f` *outputFile* `-i` *sampleInterval*

Statmonitor can be terminated by CTRL-C, or will shut down when the repository is shut down. You can use other options to stop statmonitor, or to start writing to a new file, after a given number of samples or period of time.

For complete statmonitor command line syntax, see “statmonitor” on page 413.

GemStone recommends running statmonitor at all times for production systems. If an error occurs, it may be possible to determine the cause of the problem using statmonitor data. In particular, diagnosis of problems involving memory use and GemStone repository space may require statmonitor data from before the error actually occurs. Statmonitor includes options to regularly terminate the statmonitor file and begin a new one, either by time or by the number of samples collected.

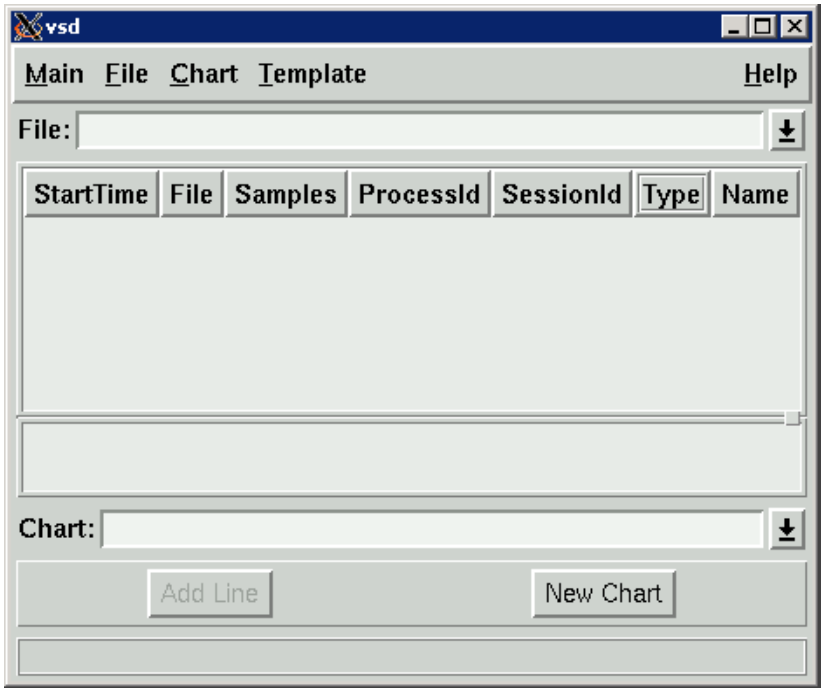
A longer sampling interval will keep file sizes smaller, but may not be fine enough granularity to catch certain types of problems.

Starting VSD

The VSD executable is in the $GEMSTONE/bin directory. To start VSD, simply execute the `vsd` command, assuming that `$GEMSTONE/bin` defined in your path.

The initial startup screen will appear similar to that shown in Figure G.1.

Figure G.1 VSD Startup Screen



Loading an existing statmonitor output file

To load a statmonitor file into VSD, do one of the following:

- To browse for an existing statmonitor output file, choose **Load Data File...** from the **Main** menu.
- If you know the name of the load file, you can type the full path in the File entry box, then press Enter.
- To switch to a statmonitor output file that you've already loaded, click the down-arrow next to the File entry box, then select a file from the list.

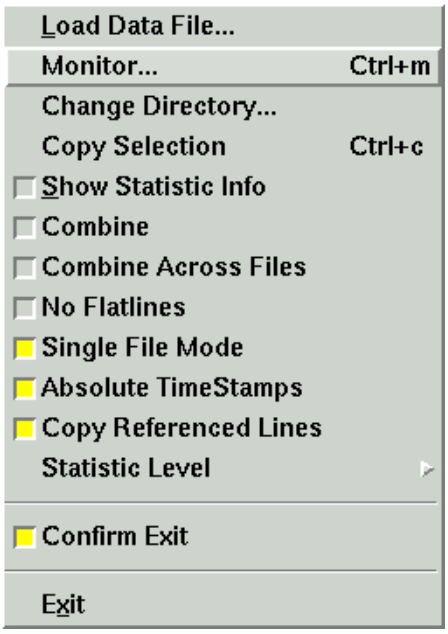
Maintaining a current view of the data file

If you select **File > Auto Update**, VSD automatically updates your display, and any associated charts, any time the data file changes. Otherwise, you can choose **File > Update** periodically to update the displays when you choose.

Starting statmonitor from VSD and viewing current data

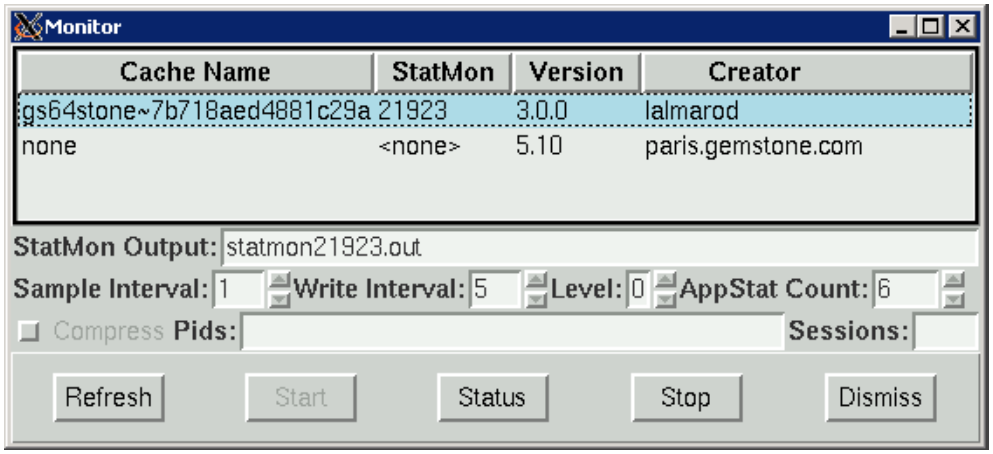
You can also use VSD to start statmonitor and read directly from the data file statmon is currently creating. To do so:

Step 1. Choose **Monitor** from the **Main** menu.



Step 2. When the Monitor window appears, click to select the cache on the local machine for which you want to obtain statistics.

Figure G.2 Monitor Window



Step 3. If necessary, modify any statmonitor startup parameters:

- The name of the statmonitor output file.
- The sample interval (in seconds)—that is, how frequently to read the cache.
- The write interval—the maximum number of seconds to wait before flushing the cached information to the output file.
- The level of application statistics to gather.
- Whether to compress the output file.
- If you want to monitor only processes with specific Process Ids or specific sessions, you may explicitly enter these. Otherwise, all processes are monitored.

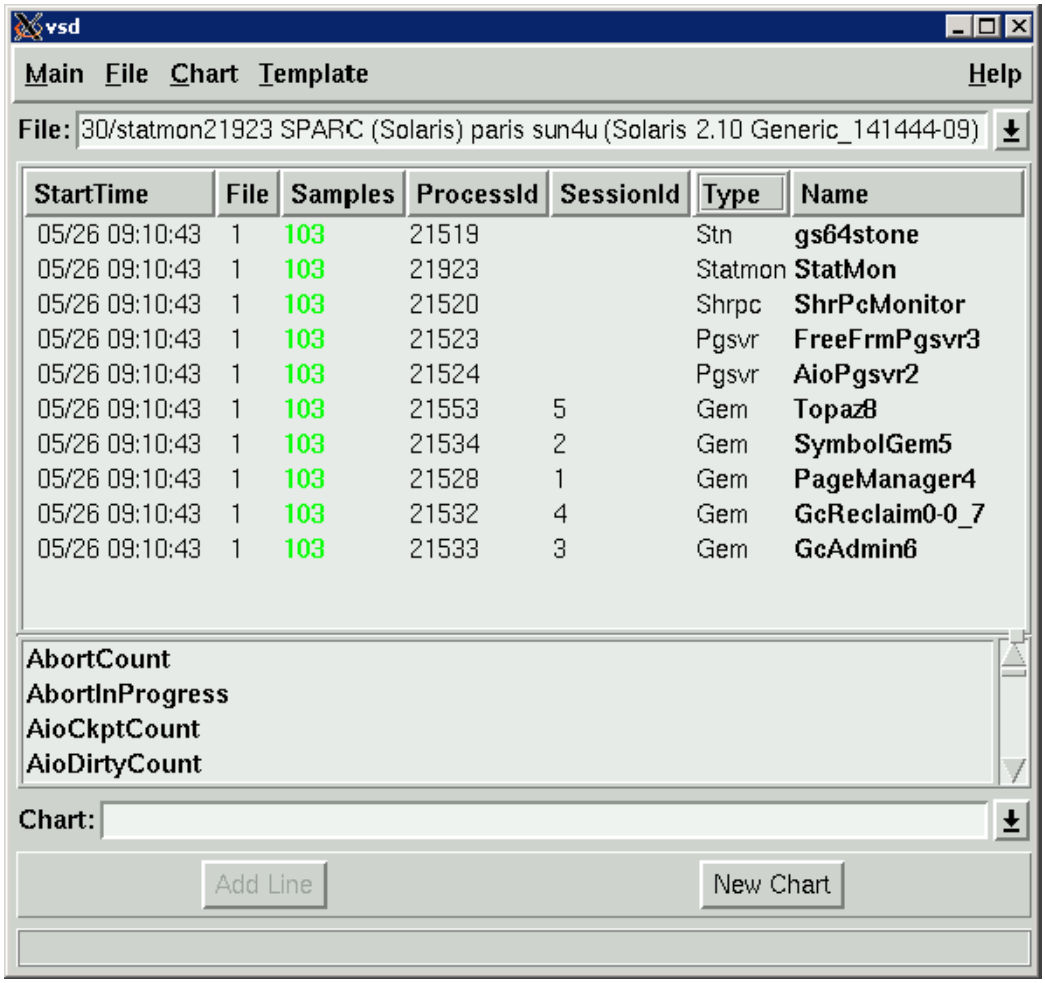
Step 4. When you're done, click **Start**.

VSD loads the data file as soon as it has some samples in it (unless you have explicitly turned off Auto Update).

Step 5. When you're finished collecting statistics, click **Stop**.

Viewing Statistics

After you've loaded the data file or started monitoring, the VSD window looks something like this:



If you are monitoring, the green numbers in the Samples column indicate processes that are still running. Black numbers indicate processes that have stopped.

Your next step is to select the specific process and cache statistics to view. There are several types of processes for which statistics are recorded. The common process types are:

Stn	Stone
Shrpc	shared page cache monitor
Pgsvr	AIO or free frame page server, or a page server for a remote session
Gem	Gem; includes system Gems, RPC Gems, and linked sessions such as linked Topaz
Statmon	statmonitor

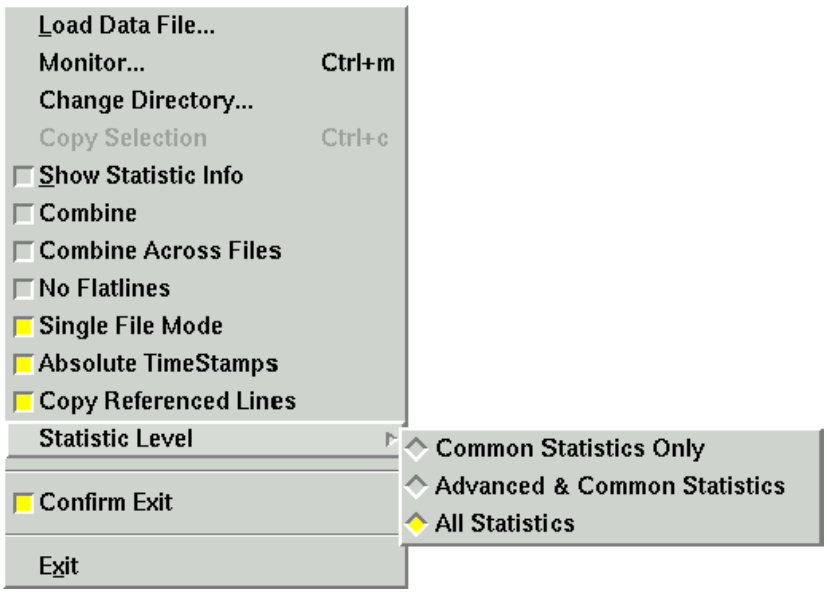
Additional types will appear if you are also monitoring OS information. These will include platform-specific statistics.

Each process type has a specific set of statistics that are available, although some statistics, such as UserTime, apply to all processes. For detailed descriptions of each statistic, including the process type, see Appendix H.

Controlling visibility of statistics

Each statistic has a characteristic called a *level* reflecting the amount of background knowledge about GemStone processes needed to use it with understanding. You can set up VSD to list (in its main window and in associated charts) only those statistics that are at, or below, a certain level of complexity.

To establish the levels of statistics that you want to display in VSD, choose **Statistic Level** from the VSD window's **Main** menu:



The level of statistics you select will be recorded in your `.vsdrc` file, and will be used automatically the next time you start VSD. See page 473 for details on VSD configuration files.

You may also hide all statistics that are flatlines — that is, the value of that statistic is 0 for the entire sample set. To hide statistics where there are no non-zero values, check **No Flatlines** in the **Main** menu or in the right mouse button popup menu.

Locating and selecting processes

By selecting a column header in the VSD display, you can sort all the processes by that column's attribute. For example, if you want to find the Stone, you can sort by Type. Or, if you know the SessionId of the Gem, you can sort by SessionId and scroll down under you find that Gem.

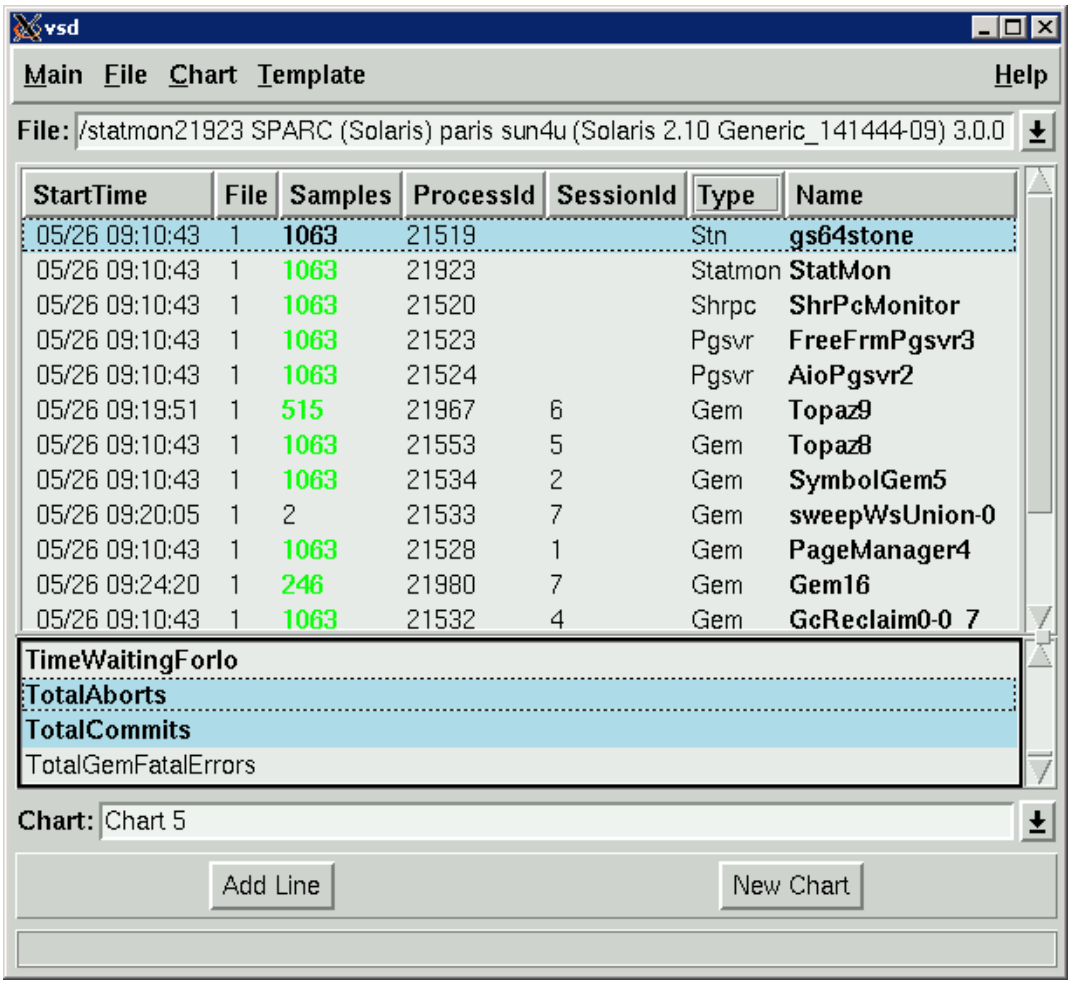
You can also search for a specific session name or process ID.

Click the mouse in the process list, then either select **search...** on the right mouse button popup menu, or press CTRL-S. When the dialog box appears, enter the PID or name of the process you're looking for. VSD highlights the first process with that PID or name.

You can make multiple selections in the list of processes. Only statistics that are common to all the selected process types will be listed. This is most useful when you want to select all processes of a particular type, such as all Gem processes or all page server processes. You can do this by multiple selection in the list, or by selecting one process of the desired type, and using the right mouse button popup menu item **Select** and then choosing **by Type**.

When you have multiple processes selected, you can either view statistical data for each process individually — which may not be practical if there are hundreds of processes — or you can combine the values. To combine values, check the value **Combine** in the **Main** menu or in the right mouse button popup menu.

Figure G.3 Process List with the Stone and Two Statistics Selected



Selecting and viewing statistics

Next, select one or more statistics in the scrolling list of available statistics for the selected processes. You can also type a letter to reach the first statistic that starts with that letter.

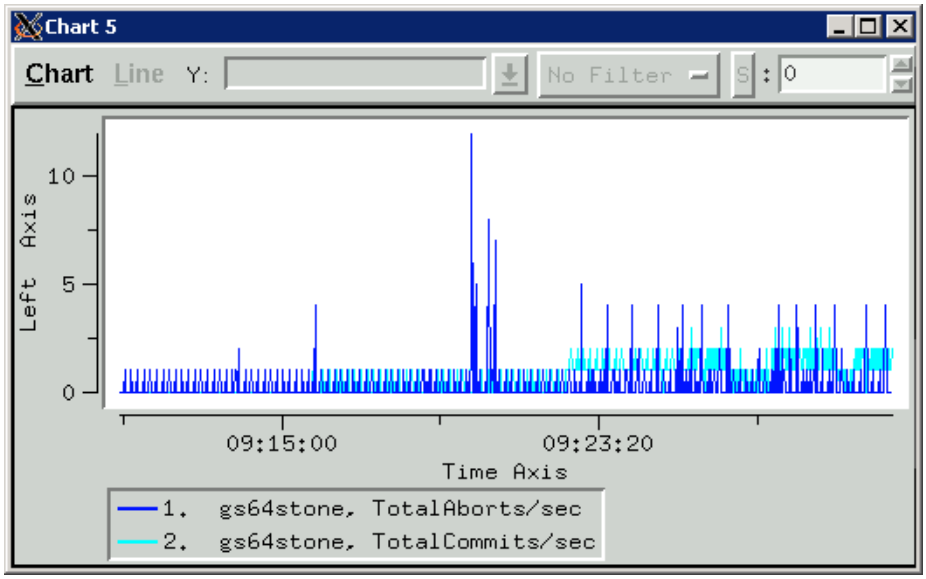
If you have selected processes of different types, only the statistics that are available for each of the processes will be displayed. Statistics that contain non-zero data points (not flatline) are shown in bold.

After you've selected the statistic, you can display the statistic in a new chart or add it to an existing chart.

To display the statistic in a new chart, click **New Chart** button. To explicitly name the chart, enter a name in the Chart entry field.

To display the statistic in an existing chart, click the down-arrow and select the chart name in the **Chart** entry box. Then click on **Add Line**. If you do not select a chart, the statistics will be displayed in the most recently selected chart.

Figure G.4 Chart Displaying Statistics



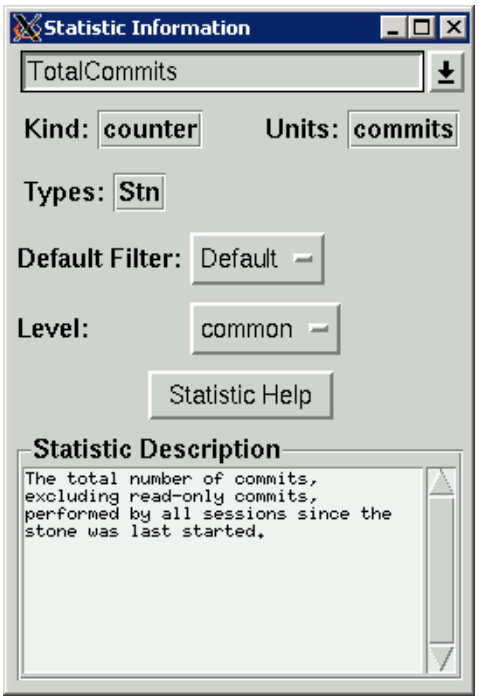
You can add more statistics, including statistics for different process types, by repeating these steps.

Customizing Your Chart

You can customize and manipulate a VSD chart in many ways:

- To select a line in the chart, click on it or on its entry in the chart legend. The line will be highlighted in red.
- To delete a line from the chart, click the middle mouse button (if available) on its entry in the chart legend. Or select the line and choose **Line > Delete**.

- To find out about a specific point in a chart, hold the mouse pointer over it.
- To view an explanation about the most recently selected statistic, go to the VSD main window and choose **Show Statistics Info** from the **Main** menu. You'll see a Statistic Information window that looks something like this:



In the Statistic Information window, you can redefine the level and default filter for any VSD statistic.

TIP

If you leave this window up as you work, it changes to reflect the current statistic; in this way you can get a quick explanation of any statistic you're currently examining.

- To aid in identifying processes, you can customize the name used for a specific Gem or Topaz session.

  To do so, log into the Gem or Topaz session as soon as possible after you've started it. At the Topaz prompt or in a GemStone workspace, evaluate:

  ```
  System cacheName: 'myName'
  ```

Filter

Whenever you add a line to a chart, the filter determines how much information is displayed for the selected statistic:

- **None** — Displays the values for the statistic as they are expressed in the statmonitor text file.
- **PerSample** — Displays the difference between two consecutive samples of the statistic.
- **PerSecond** — Displays the difference between two consecutive samples of the statistic, divided by the number of elapsed seconds between the two samples.
- **Aggregate** — Displays a running total for the statistic. Each raw value from previous samples is added to the current one.

Once you've added the line to a chart, you can override its default filter by specifying a new filter from the drop-down menu at the top of the Chart window.

Using VSD Chart Templates

VSD templates let you quickly add a set of lines to a chart. Templates are helpful if you find yourself performing the same task frequently in VSD — for example, monitoring the same five or six statistics. By creating a template for the statistics that you want to monitor most frequently, you can automate the task of building charts.

In your template, you can assign a filter for each statistic, to determine how much information is displayed for that statistic. You can also restrict the template to look for extreme conditions (for example, Gem processes that are consuming 90% or more of the CPU).

VSD is shipped with a set of predefined templates, maintained in the `.vsdtemplates` file in your home directory.

NOTE

You can use a text editor to change or delete templates in the `.vsdtemplates` *file. For details about the format of the* `.vsdtemplates` *file, see* *"VSD Files" on page 473.*

To do this...	Do this...
Create a new chart from a template	In the VSD main window, choose **New from Template** from the **Chart** menu. This is a good way to display some of the more useful system statistics.
Apply a template to the chart that you're viewing	In the Chart window, choose **Add From Template** from the **Chart** menu. *NOTE* *If you have zoomed in on a chart, the template filter is applied only to values within the zoomed range.*
Reread the template file `.vsdtemplates` into VSD after you've edited it	In the VSD main window, choose **Reload Template File** from the **Chart** menu.
Save the current chart as a template	In the Chart window, configure the chart as you desire, then choose **Save Template** from the **Chart** menu. When you exit VSD, the template will be saved to the `.vsdtemplates` file. If you save the current chart as a template, you may still need to edit the `.vsdtemplates` file so that you can get the selected Gem or page server.

G.2 VSD Menu Reference

Chart menu

To customize the way VSD displays statistics in your chart, choose items from the Chart window's **Chart** menu.

- **Zoom In** — You can zoom to improve your view of the chart. After you choose this menu item, click to select one corner of the area that you want to zoom. Move the mouse pointer to the opposite corner of the zoom area, then click again.

  To quickly zoom in on an area, move the mouse pointer over the area and click the middle mouse button (if available).

- **Zoom Out** — Select this menu item or click the right mouse button to reverse the effect of one zoom-in operation.

- **Compute Scale All, Unscale All** — To view multiple statistics on the same axis, it's sometimes helpful to adjust the scale of the chart.
- **Show Legend** — Select this item to display the legend for this chart.
- **Time Format** — Changes the format of the time displayed along the X axis.
- **Show Time Axis Title, Show Left Axis Title, Show Right Axis Title** — Select these items to display the title alongside the respective axes.
- **Show Current Values** — Select this item to display the current X and Y values for the selected line at the top of the chart.
- **Show Min and Max** — Select this item to display the minimum and maximum values for the selected line at the top of the chart.
- **Show Line Stats** — Select this item to display the following statistics for the selected line: the number of data samples, the minimum, the maximum, the mean, and the standard deviation. The statistics are calculated from all of the data points on the selected line in the region defined by the graph's current X axis. (To change the region, use the **Zoom In** or **Zoom Out** menu item.)
- **Show CrossHairs** — Select this item to display crosshairs centered at the current cursor position, useful to precisely locate a point of interest in time or on the vertical scaled.
- **Show Gridlines** — Select this item to display gridlines in the chart background, also useful for precisely locating a point of interest.

For additional information about what you can do in the Chart window, choose **Help** from the **Chart** menu.

Line menu

Line menu commands operate on the currently selected line. To select a line, click on it or on its entry in the chart legend. VSD highlights the selected line.

- **Log Info** — Displays a log file showing the line statistics for all data samples in the region defined by the graph's current X axis.
- **Log Delta** — Use this menu item to measure the difference between two values on the selected line.

  Select the line, choose this menu item, then click on the two points whose difference you want to compute. VSD responds by displaying a log file showing the difference in time and value between the two points; the number of data samples in the selected line segment; and the minimum, maximum, mean, and standard deviation of those samples.

- **Compute Scale** — Computes a scale value for the selected line o make it visible on the current chart. You can also use the Scale entry box to manually change the scale. The default scale value is 1.
- **Unscale** — Reverses the effect of **Compute Scale**.
- **Graph on Left Axis** — Select this item to display the Y axis for the selected line to the left of the chart. Otherwise, the Y axis is displayed to the right.

  To view multiple lines on the same chart, you can use this menu item to graph large values on one axis and small values on the opposite axis.
- **Symbol** — Selects a new symbol.
- **Update** — Updates the selected line, assuming that values are being monitored in real time and that **Auto Update** is turned off on the main window's File menu.
- **Delete** — Removes the selected line from this chart.

G.3 VSD Files

VSD writes the following files to your home directory. This information is useful primarily to users who want to do more complex configuration or work directly with template files.

.vsdrc

This file is used by VSD to record its configuration. VSD reads the file when it starts and writes the file when it exits.

If you delete the `.vsdconfig` file, then the next time VSD is started, the file will be recreated with default values.

Do not modify this file manually, since it will be overwritten and your changes lost. If you want to specify configuration values that will be retained from one session to the next, do so in `.vsdconfig` instead.

.vsdconfig

You can configure VSD by setting default values in this file. Any value set in `.vsdconfig` will override the same definition in `.vsdrc`.

If you delete the `.vsdconfig` file, then the next time VSD is started, the file will be recreated with default values.

.vsdtemplates

This file contains VSD chart templates. A template is a list of line specs that define the content and appearance of a named VSD chart. For example, the following is a template for displaying a chart with information that you can use to determine if your Shared Page Cache is too small:

```
set vsdtemplates(CacheTooSmall) {
        {Shrpc ShrPcMonitor FreeFrameCount 1 none y2}
        {+ {*} FramesFromFreeList 1 persecond y}
        {+ {*} FramesFromFindFree 1 persecond y}
    }
```

Each line spec has the following format:

```
   {type name stat scale filter axis [statFilter]}
```

type — One of the following: Stn, Shrpc, Gem, or Pgsvr. If an optional '+' is appended to the type name, all processes that match this line spec will be combined into a single line.

name — Identifies the specific process. You can use a pattern identifier.

stat — A valid counter name.

scale — A number.

filter — One of the following: none, persecond, persample, or aggregate.

axis — Either y or y2.

statFilter (optional) — A list that lets you exclude lines from the chart, based on the values of their statistical counters. Its format is:

```
      {count min max mean stddev}
```

For examples, examine the predefined templates in .vsdtemplates. For more about VSD templates and the syntax and options available, see the VSD help under the topic "Template Syntax".

Appendix

H Cache Statistics

This section lists the GemStone/S 64 Bit statistics in alphabetical order. The heading indicates the processes for which the statistic is applicable: Stone, Gem, SPC (shared page cache) monitor, Pgsvr (AIO page server), or all.

NOTE

Certain statistics not listed in this chapter, but visible in VSD displays, are for internal purposes only.

AbortCount (Gem)

The number of aborts executed by a Gem process (or by an application linked to a Gem) since the Gem was most recently started.

AbortInProgress (Gem)

A boolean that indicates if the session is currently performing an abort operation.

AioCkptCount (Pgsvr)

The number of dirty pages written from the shared page cache to disk to satisfy a checkpoint.

AioDirtyCount (Pgsvr)

The number of dirty pages written from the shared page cache to disk by Stone's AIO page server during normal operation.

AioNumBuffers (Stone)

The current setting of STN_MAX_AIO_REQUESTS

AioNumEmptyBuffers (Stone)

The number of internal AIO buffers that are currently empty. The stone will block when starting a tranlog write unless at least 3 buffers are empty.

AioRateLimit (Pgsvr)

The current I/O rate being used by the page server, expressed in I/O operations per second. The page server will perform no more than this number of I/O operations per second on average.

AioRateMax (Pgsvr)

The current maximum I/O rate being used by the page server, expressed in I/O operations per second. The page server will perform no more than this number of I/O operations per second on average.

AioWriteFailures (Pgsvr)

Total number of write errors detected by this page server, including write errors that succeeded on retry.

AsyncFlushesInProgress (Pgsvr)

The number of pending extent flush operations.

BackupHighWaterPage (Stone)

The lowest page ID for which the GC reclaim gems will temporarily ignore because a full backup is in progress. Pages with IDs less than this value will be reclaimed normally. Page IDs equal to or greater than this value will not be reclaimed. The session performing the full backup will increase this value as the full backup progresses.

BackupRecordPagesWrittenByGem (SPC monitor)
BackupRecordPagesWrittenByStone (SPC monitor)

The number of backup record pages in the cache because of a write done by a Gem or by the Stone, respectively.

BitlistPagesWrittenByGem (SPC monitor)
BitlistPagesWrittenByStone (SPC monitor)

The number of bitlist pages in the cache because of a write done by a Gem or by the Stone, respectively. A bitlist page is a form of a bit array.

BitmapPageReads (all)

The number of bitmap pages read by the process since it was last started. These page reads are actual disk reads and not reads from the shared page cache.

BmCHeapPages (Gem, Stone)

The number of temporary bitmap pages allocated on the C heap of the process.

BmInternalPagesWrittenByGem (SPC monitor)
BmInternalPagesWrittenByStone (SPC monitor)

The number of bitmap internal pages in the cache because of a write done by a Gem or by the Stone, respectively.

BmLeafPagesWrittenByGem (SPC monitor)
BmLeafPagesWrittenByStone (SPC monitor)

The number of bitmap leaf pages in the cache because of a write done by a Gem or by the Stone, respectively.

CacheScanCount

Number of times the shared page cache monitor statistics thread has scanned the entire shared page cache.

CacheSlotIndex (all)

Index of the slot in the shared page cache used by this process.

CacheStartQueueSize (Stone)

The number of sessions waiting for remote cache startup.

CHeapSizeKB (all)

The size (in KB) of the C heap area; that is, the total size of results of UtlMalloc for which UtlFree has not been called. This statistic is computed only in slow or fastdebug executables.

CheckpointCount (Stone)

The number of checkpoints that have been written since the Stone repository monitor was last started. Writing a checkpoint implies that all of the data and meta information needed to recover the data corresponding to the commit record associated with the checkpoint have been written to the disk(s) containing the extent(s) that make up the repository. Thus, the last checkpoint in the transaction log determines how much data in the log must be recovered when there is a system crash.

In full logging mode, the checkpoints are controlled completely by the STN_CHECKPOINT_INTERVAL configuration parameter (page 376). In partial logging mode, a checkpoint may be written more often if the size of the transaction exceeds the value set by the configuration parameter STN_TRAN_LOG_LIMIT (page 391). If partial logging is in use, a rapidly increasing CheckpointCount indicates that STN_TRAN_LOG_LIMIT may be set too small.

CheckpointState (Stone)

This internal statistic indicates the state of the checkpoint process:

0 — checkpoint not active
1 — AIO servers writing pages
2 — pre-fsync processing
3 — synchronous fsync
4 — asynchronous fsync
5 — second fsync
6 — dispose alloc pages shadowed
7 — write log record

These values are provided for information only and are subject to change without notice.

ClassesRead (Gem)

The number of classes copied into VM memory since the start of session.

CleanSlotsInRecoveryCount (SPC monitor)

The current number of slots being recovered which were owned by a process which shutdown cleanly.

CleanSlotsRecoveredCount (SPC monitor)

The total number of slots the shared cache monitor has recovered because a client process shutdown cleanly.

CleanSlotThreadTimeRunningMs (SPC monitor)

Approximate real time in milliseconds that the clean slot recovery thread in the shared page cache monitor has spent doing work.

ClientAbortInProgress (Pgsvr)

A boolean indicating if the page server's client Gem is currently performing an abort operation.

ClientLostOtsSent (Pgsvr)

The number of Lost OT root signals sent from the stone to the page server's client.

ClientPageReads (Pgsvr)

The number of pages that have been transmitted by a page server to its client. This statistic is implemented only for cache slots used by a page server.

ClientPageWrites (Pgsvr)

The number of pages that have been transmitted by a client to its page server. This statistic is implemented only for cache slots used by a page server.

ClientPid (Pgsvr)

The process ID of the client process associated with this AIO page server process.

ClientSigAbortsSent (Pgsvr)

The number of SigAborts sent from the stone to the page server's client.

ClockHandFrameId (Pgsvr)

The shared page cache frame ID that the page server clock hand currently references.

CodeCacheSizeBytes (Gem)

The total size in bytes of copies of GsNMethods that are in the code generation area and ready for execution, as of the end of mark/sweep.

CodeGenGcCount (Gem)

The number of times the code generation area has been garbage collected.

CommitCount (Gem)

The number of commits executed by a Gem process (or application linked to a Gem) since the Gem was most recently started.

CommitQueueAddedToRunQueueCount (Stone)

The number of times sessions from the commit queue were added to the run queue.

CommitQueueAddedToRunQueueSessionCount (Stone)

The number of sessions from the commit queue which were added to the run queue.

CommitQueueSessionNotReadyCount (Stone)

The number of times the stone was ready to assign the commit token to a session in the commit queue but no session was not ready to receive the token.

CommitQueueSize (Stone)

The number of Gem session processes waiting for the commit token.

CommitQueueThreshold (Stone)

The current setting of the STN_COMMIT_QUEUE_THRESHOLD configuration parameter. This setting determines how large the commit queue must be before the stone will defer commit record disposal. A value of -1 indicates this feature is disabled and commit record disposal will never be deferred.

CommitRecordCount (Stone)

The number of outstanding commit records that are currently being maintained by the system. A number larger than the STN_SIGNAL_ABORT_CR_BACKLOG configuration option (page 390) indicates that there is a process in a transaction that is preventing the Stone from reclaiming (garbage collecting) the resources

associated with those commit records. Large values are usually accompanied by continuing growth in the size of the repository.

CommitRecordDisposalsDeferredCount (Stone)

The number of times the stone deferred commit record disposal because the number of sessions in the commit queue exceeded the STN_COMMIT_QUEUE_THRESHOLD setting.

CommitRecordDisposalState (Stone)

An internal statistic used to analyze the commit record disposal routines.

CommitRecordPagesWrittenByGem (SPC monitor)
CommitRecordPagesWrittenByStone (SPC monitor)

The number of commit record pages in the cache because of a write done by a Gem or by the Stone, respectively.

CommitRecordsDisposedCount (Stone)

The total number of commit records disposed by the Stone.

CommitRecordsReadAbortCount (Gem)

The number of commit records read while processing an abort.

CommitRecordsReadCommitBeforeCommitQueueCount

The number of commit records read by the session before it requested the commit token during a commit operation.

CommitRecordsReadCommitInCommitQueueCount

Number of commit records read by the session after it has requested but not yet received the commit token.

CommitRecordsReadCommitWithTokenCount (Gem)

The number of commit records read while processing a commit and the Gem has the commit token.

CommitRetryFailureCount (Gem)

The number of commits that failed after exceeding the retry count.

CommitsSinceLastEpoch (Stone)

Number of commits, excluding reclaim commits by GcGems, since the start of the currently open epoch.

CommitTokenSession (Stone)

The session ID of the gem holding the commit token. If no gem is holding the commit token, the value will be zero.

CountBagInteriorPagesWrittenByGem (SPC monitor)
CountBagInteriorPagesWrittenByStone (SPC monitor)

The number of count bag interior pages in the cache because of a write done by a Gem or by the Stone, respectively.

CountBagLeafPagesWrittenByGem (SPC monitor)
CountBagLeafPagesWrittenByStone (SPC monitor)

The number of count bag leaf pages in the cache because of a write done by a Gem or by the Stone, respectively.

CountMapLeafPagesWrittenByGem (SPC monitor)
CountMapLeafPagesWrittenByStone (SPC monitor)

These statistics are obsolete; the page kind is no longer used.

CrashedSlotsInRecoveryCount (SPC monitor)

The current number of slots being recovered which were owned by a crashed process.

CrashedSlotsRecoveredCount (SPC monitor)

The total number of slots the shared cache monitor has recovered because a client process shutdown abnormally.

CrashedSlotThreadTimeRunningMs (SPC monitor)

Approximate real time in milliseconds that the crashed slot recovery thread in the shared page cache monitor has spent doing work.

CrPageLocateForDisposeCount (Stone)

Number of times a commit record page was not found in the stone's commit record cache at commit record disposal time.

DataPageReads (all)

The number of data pages read by the process since it was last started. These page reads are actual disk reads and not reads from the shared page cache.

DataPagesWrittenByGem (SPC monitor)
DataPagesWrittenByStone (SPC monitor)

The number of data pages in the cache because of a write done by a Gem or by the Stone, respectively.

DeadDataPagesWrittenByGem (SPC monitor)
DeadDataPagesWrittenByStone (SPC monitor)

The number of dead data pages in the cache because of a write done by a Gem or by the Stone, respectively. Dead data pages are intended for disposal.

DeadNotReclaimedKobjs (Stone)

The number of objects that have been determined to be dead (current sessions have indicated they do not have a reference to these objects) but have not yet been reclaimed. (Units: 1000s of objects.)

DeadObjsReclaimedCount (Stone)

The total number of dead objects reclaimed since the Stone repository monitor process was last started. For a system in "steady state" for a particular application, look for a uniform discovery rate per garbage collection epoch. Increasing the duration of the epoch should increase this value, but that could also cause larger swings in the amount of free space in the repository.

DeferCkptCompleteCount (Stone)

The number of pending commit operations for which the checkpoint will allow itself to be deferred before it completes.

DepMapKeysChanged (Gem)

The total number of objects with DependencyMap changes committed by this session.

DirtyListSize (Gem)

The number of modified committed objects in the temporary object memory dirty list.

DirtyPageSweepCount (Pgsvr)

The number of times the page server has swept the cache for dirty pages or frames to add to the free list.

EmptyBmLeafPagesWrittenByGem (SPC monitor)

The number of empty bitmap leaf pages in the cache because of a write done by a Gem.

EmptyBmLeafPagesWrittenByStone (SPC monitor)

The number of empty bitmap leaf pages in the cache because of a write done by the Stone.

Env1sendCacheSerialNum

Counter on env 1 send site caches.

EpochForceGc (Stone)

This statistic is 1 when a user has requested that Epoch garbage collection be started manually, and is otherwise zero.

EpochGcCount (Stone)

The number of times that the Epoch garbage collection process was run by the Admin GcGem since the Admin GcGem was started. For a system in steady state, look for uniform periods between runs or a uniform run rate.

EpochLastDuration (Stone)

The duration of the last completed epoch garbage collection operation in seconds.

EpochNewObjs (Stone)

The number of new objects that were created during the last epoch.

EpochPossibleDeadObjs (Stone)

The number of possible dead objects found by the last Epoch garbage collection.

EpochScannedObjs (Stone)

The number of objects scanned by the last Epoch garbage collection.

ExportedSetSize (Gem)

The number of objects in the ExportSet. The ExportSet is a collection of objects for which the Gem process has handed out an Oop to GemBuilder or Topaz. Objects in the ExportSet are prevented from being automatically garbage collected in the Gem's temporary object memory (see page 313). The ExportSet is used to guarantee referential integrity for objects only referenced by an application, that is, objects that have no references to them within the Gem.

The application program is responsible for timely removal of objects from the ExportSet. The contents of the ExportSet can be examined using hidden set methods defined in class System. In general, the smaller the size of the ExportSet, the better the performance is likely to be. There are a couple of reasons for this relationship. The ExportSet is one of the root sets used for garbage collection. The larger the ExportSet, the more likely it is that objects that would otherwise be considered garbage are being retained. One threshold for performance is when the size of the export set exceeds 2K objects. When its size is smaller than 2K objects, the export set is stored as a single disk page. When its size is larger than 2K, the export set occupies more than one page and is likely to cause additional I/O.

ExportedSetSizePinnedInMemory (Gem)

The number of objects in the ExportSet that cannot drop out of temporary object memory nor be garbage collected by in-memory collection of temporary objects. Will include any objects represented in ExportedSetSize that are not committed. Can be less than ExportedSetSize if ((System gemConfigurationAt:GemDropCommittedExportedObjs) == true).

ExtentFlushCount (all)

The cumulative number of file flush operations performed on any extent by the process. Note that extents residing on raw partitions do not require flushing. On UNIX systems, file flushes are performed by calling the fsync() function. During a checkpoint, each extent is flushed once, except for the primary extent which is flushed twice. Most extent flushes are performed by the AIO page servers.

FailedCommitCount (Gem)

The number of attempts to commit that failed due to concurrency conflicts.

FragmentBmPagesWrittenByGem (SPC monitor)
FragmentBmPagesWrittenByStone (SPC monitor)

The number of bitmap fragment pages in the cache because of a write done by a Gem or by the Stone, respectively.

FrameCount (SPC monitor)

The size of the shared cache in frames.

FramesAddedToFreeList (all)

The number of frames added to the free list since the session (for Gems), shared page cache, or Stone started.

FramesFromFreeList (all)

The number of times the process acquired a page frame in the shared page cache from the list of free frames.

FramesFromFindFree (all)

The number of times the process acquired a page frame in the shared page cache by scanning the cache entries. The process tries to find free frames this way instead of taking them from the free list when the number free is below the value set by the GEM_FREE_FRAME_LIMIT configuration option (page 363). While scanning for free frames under those conditions is desirable from a system perspective, it represents additional overhead for the particular session.

FreeFrameCacheNumFrames (all)

The number of frames present in the free frame cache for the process.

FreeFrameCacheSize (all)

The number of frames that the process will add or remove from the shared free frame list in a single operation. A value of zero indicates that free frames are not cached and will be added or removed from the shared free frame list one at a time.

FreeFrameCount (SPC monitor)

The number of unused page frames in the shared page cache. This statistic gives some indication of the utilization of the cache, but it is not tunable. This statistic is valid (non-zero) only for the shared page cache monitor process slot.

FreeFrameLimit (all)

When the number of free frames in the shared page cache is less than the FreeFrameLimit, the Gem scans the cache for a free frame rather than use one from the free frame list, so that the Stone process can use the remaining free frames.

FreeOops (Stone)

The number of free OOPs in the free list that have not been allocated to a Gem *and* committed.

FreePages (Stone)

The size of the free page pool for the repository. Free space in the repository is calculated at 16 KB for each page in the free pool.

GarbageCollectionState (Gem)

The state of a garbage collection task in the Admin GcGem or a user performing a garbage collection operation. Values are defined as follows:

0 - Garbage collection not active
1 - Initialization
2 - Mark-sweep
3 - Compute dead objects
4 - Remove not dead objects
5 - Finalize weak references
6 - Start record dead
7 - Write possible dead objects to tranlog
8 - Finish record dead
9 - Write not dead objects to tranlog
10 - Send not dead objects to stone

GcHighWaterPage (Stone)

The lowest page ID for which the GC reclaim gems will temporarily ignore because a garbage collection operation is in progress. Pages with IDs less than this value will be reclaimed normally. Page IDs equal to or greater than this value will not be reclaimed. The session performing the garbage collection will increase this value as the operation progresses

GciBytesRcvd (Gem)

Number of bytes received from the RPC GCI client.

GciBytesSent (Gem)

Number of bytes sent to the RPC GCI client.

GciPhysBytesRcvd (Gem)

Number of bytes received from the RPC GCI client before decompression.

GciPhysBytesSent (Gem)

Number of bytes sent to the RPC GCI client after compression.

GciRpcCommandsServiced (Gem)

The number of GCI RPC commands serviced by this Gem.

GciRpcKeepAlivePacketCount (Gem)

Number of keep-alive packets received from the GCI client on a remote host.

GcLockKind (Stone)

Indicates the state of the garbage collection lock and why it is being held:

0 - Free
1 - MarkForCollection
2 - FindDisconnectedObjects
3 - Pre-MarkGcCandidates (loading candidate objects)
4 - MarkGcCandidates
5 - Epoch GC
6 - Reclaim All
7 - (not used)
8 - Repository Scan (listInstances, listReferences, etc)

GcVoteState (Stone)

GcVoteState tracks the progress of a number of phases within garbage collection. Possible values are:

- 0 – Not voting
- 1 – Gems are voting on the possible dead set
- 2 – Vote completed, awaiting start of Possible Dead Write-Set Union Sweep (PDWSUS).
- 3 – PDWSUS in progress

- 4 – PDWSUS completed

Note that all of these phases must complete before the PossibleDead objects can be "promoted" to DeadNotReclaimed objects. The Admin GcGem is responsible for performing the possible dead write set union sweep, and must be running for this to occur.

For details about the voting phase of garbage collection, see "What Happens to Garbage?" on page 309.

GcWsUnionSize (Stone)

The number of objects in the write set union used to finalize possible dead objects. All objects in the WSU must be scanned to determine if any objects reference one or more possible dead objects.

GcWsUnionSweepCount (Stone)

The total number of sweeps of the possible dead write set union that have been done by the Admin GcGem since it started.

GemFreePages (Gem)

The number of free pages that the VM or Gem has acquired but has not yet used.

GemHasCommitToken (Gem)

A boolean that indicates whether the Gem holds the commit token.

GemsInCacheCount (SPC monitor)

The total number of Gems using the shared page cache whose process slot you are viewing. This is useful for distinguishing Gems using a local shared page cache from those using a remote shared page cache.

GemTempObjCacheSizeKb (Gem)

The value of the session's GEM_TEMPOBJ_CACHE_SIZE_KB configuration parameter.

GlobalDirtyPageCount (SPC monitor)

The total number of pages in the shared cache that are dirty but not yet eligible for asynchronous writing to the disk because they have not yet been committed. If this value is very large, then very large transactions may be filling the cache. Otherwise, if the Stone repository monitor is running on this cache, the Stone's

private page cache size may be too small. This statistic is available only for the shared page cache monitor's slot (currently Slot 0).

GlobalStat00 — GlobalStat47 (Stone)

User-defined statistics that can be written and read by any Gem on any Gem server; that is, these statistics are stored in the shared page cache of the Stone, rather than that of the machine on which the Gem is running. There are 48 global cache statistic slots available. For details, see "Global Session Statistics" on page 179.

GsMsgCount (Stone)

The number of messages processed by the Stone on behalf of Gems. This statistic can help determine how busy the Stone is.

GsMsgKind (Stone)

An integer identifying the kind of message the Stone is currently processing.

GsMsgSessionId (Stone)

The session identifier of the Gem for which the Stone is currently processing a message.

IndexProgressCount (Gem)

Used to monitor the status of certain index operations:

0 - Inactive
1 - Identity index audit in progress.
2 - Equality index audit - auditing root terms.
3 - Equality index audit - auditing NSC counts.
4 - Equality index audit - auditing btree counts.
5 - IndexManager>>removeAllIndexes in progress.

InvalidPagesWrittenByGem (SPC monitor)
InvalidPagesWrittenByStone (SPC monitor)

The number of invalid (i.e. empty) pages in the cache because of a write done by a Gem or by the Stone, respectively.

LastCommandFromClient (Pgsvr)

The numeric index of the last message sent to the page server by its client.

LastErrorNumber (Gem)

The number of the last error processed by the session. Fatal errors are not reported in this statistic.

LastSessionFatalError (Stone)

The session ID of the last session that reported a fatal error to the stone.

LastSessionIdStopped (Stone)

The session ID of the last session that was sent a stop session message.

LastSessionIdTerminated (Stone)

The session ID of the last session that was forcibly logged off by the stone.

LastSessionLostOt (Stone)

The session ID of the last session that was sent a SigLostOtRoot signal.

LastSessionSigAbort (Stone)

The session ID of the last session that was sent a SigAbort.

LastSleepTimeBetweenScans (SPC monitor)

Number of milliseconds the shared page cache monitor statistics thread slept after the last complete scan of the cache.

LastSleepTimeWithinScan (SPC monitor)

Number of milliseconds the shared page cache monitor statistics thread last slept while in the loop that computes cache statistics. The thread will sleep for a short interval each time it scans 100,000 cache frames.

LdiThreadOperations (Stone)

The number of wakeups from semaphore wait in stone NetLdiCall Thread.

LocalCacheAllocatedPceCount (Gem, Stone)

The number of page cache entries that the process has allocated for collision chains.

LocalCacheFreeFrameCount (Gem, Stone)

The number of frames in the private page cache that are free.

LocalCacheFreePceCount (Gem, Stone)

The number of page cache entries in the free pool.

LocalCacheOverflowCount (Gem, Stone)

The number of times that a page was moved from private cache to the shared cache because the private cache was full.

LocalCachePceCountLimit (Gem, Stone)

The maximum number of page cache entries that the process will allocate before performing a reclaim.

LocalCachePceReclaimCount (Gem, Stone)

The number of page cache entry reclaim operations performed by the process.

LocalCacheStalePcesRemovedCount (Gem, Stone)

The number of stale page cache entries that the process has removed from its private page cache lookup table. Stale PCEs occur when a reference to page in the shared cache becomes invalid because the page is removed from the cache by another process.

LocalCacheValidPcesRemovedCount (Gem, Stone)

The number of valid page cache entries removed by a reclaim operation.

LocalDirtyPageCount (SPC monitor)

The total number of pages in the shared cache that are dirty and eligible for asynchronous writing to the disk. The Stone's AIO page server will write these pages to the disk. This statistic is available only for the shared page cache monitor's slot (currently Slot 0).

LocalPageCacheHits (all)

The number of times a page lookup found the page in the shared page cache. No I/O was required to access the page.

LocalPageCacheMisses (all)

The number of times a page was not found in the shared cache and a read operation was required to get the page.

LockReqQueueSize (Stone)

The number of Gem session processes waiting for a commit to complete so that their lock request can be serviced.

LockWaitQueueSize1 — LockWaitQueueSize10 (Stone)

The number of sessions waiting for the Application object lock with the given number. (System waitForApplicationWriteLock:queue:autoRelease:)

LogHighQueueAdds (Stone)

Number of additions to the LogHighQueue. See LogHighQueueSize.

LogHighQueueSize (Stone)

The number of sessions waiting for transaction log writing to be not saturated. This queue holds mostly committing sessions

LoginQueueSize (Stone)

The number of sessions waiting for login completion.

LoginRequestsCount (Stone)

The total number of login requests processed since the Stone started.

LogLowQueueAdds (Stone)

Number of additions to the LogLowQueue. See LogLowQueueSize.

LogLowQueueSize (Stone)

The number of lower priority sessions waiting for transaction log writing to be not saturated. This queue holds primarily sessions logging resourceIds such as possibleDead or notDead.

LogRecordPagesWrittenByGem (SPC monitor)
LogRecordPagesWrittenByStone (SPC monitor)

The number of transaction log record pages in the cache because of a write done by a Gem or by the Stone, respectively.

LogRecordsIoCount (Stone)

The number of physical write operations performed on the transaction logs since the Stone repository monitor process was last started. The minimum write to a transaction log is 512 bytes (one log record). The maximum number of bytes written in a single I/O to the transaction log is 64K. The implication for performance tuning is that to achieve the best throughput (in transactions per second) you would like to have as few as possible writes to the transaction logs. The technique for achieving this is to tune the size of the transactions so that each transaction writes one or more completely filled 64K records.

LogRecordsWritten (Stone)

The number of log records that have been written to the transaction logs since the Stone repository monitor process was last started. The size of a log record is 512 bytes.

LogWaitQueueSize (Stone)

The size of the queue that holds sessions waiting for space to become available in a transaction log. This queue should be empty or nearly so unless the space for logging transactions has been exhausted.

LookupCacheSerialNum

Counter on per-class method lookup caches.

LostOtDeferCount (Stone)

The total number of times Stone has deferred sending a LostOtRoot signal to a session.

LostOtPagesWrittenByGem (SPC monitor)
LostOtPagesWrittenByStone (SPC monitor)

The number of lost OT pages in the cache because of a write done by a Gem or by the Stone, respectively.

LostOtsReceived (Gem)

The number of Lost OT Root signals received and recognized by this session.

LostOtsSent (Gem)

The number of Lost OT Root signals the Stone has sent to this session, although the session may be in a sleep or I/O wait state and not yet aware of having received the signal. (See LostOtsReceived (Gem), above.)

MainThreadTimeRunningMs (SPC monitor)

Approximate real time in milliseconds that the main shared page cache monitor thread has spent doing work.

MarkSweepCount (Gem)

The number of mark/sweeps executed by the in-memory garbage collector.

MaxVotingSessions (Stone)

Maximum number of sessions that can concurrently vote on possible dead objects at the end of an MFC (`markForCollection`) or Epoch garbage collection.

MeSpaceAllocatedBytes (Gem)

The number of bytes allocated for remSet, in-memory oopMap, and map entries.

MeSpaceUsedBytes (Gem)

The number of bytes occupied by remSet, in-memory oopMap, and in-use map entries.

MessageKindToStone (Gem)

The message type of the most recent message sent to the Stone.

MessagesToStnProcessingCommit (Gem)

The number of messages sent to the Stone while the gem is processing its part of the commit.

MessagesToStnStoneCommit (Gem)

The number of messages sent to the Stone while the Stone is processing its part of the commit.

MessagesToStnWaitingForCommit (Gem)

The number of messages sent to the Stone while waiting for the commit token.

MessagesToStone (Gem)

The number of messages sent by the Gem session process to the Stone repository monitor using shared memory as the channel.

MethodsRead (Gem)

The number of GsNMethods copied into VM memory since the start of this session.

MilliSecPerIoSample (all)

MilliSecPerIoSample is used as a parameter to implement the configurable I/O limit for GemStone processes. Because a process's I/O rate currently is sampled every five I/O operations, this statistic is computed as 1000/(ioLimit * 5). A value of 1 means that the process has no I/O limit. If the time in milliseconds since the last sample equals or exceeds MilliSecPerIoSample, the process can perform another I/O operation; if not, the process sleeps. MilliSecPerIoSample is particularly useful in limiting I/O rate of a process that is executing a long-running operation. If you want to calculate the current I/O limit, it is given by 1000/(MilliSecPerIoSample * 5).

MIClearAllCount (Gem)

Number of times that all send-site caches were cleared.

MIFullLookupCount (Gem)

Number of times that fullMethodLookup was called.

MILuCacheGrowCount (Gem)

Number of times that a per-class lookup cache was grown to larger table size.

MILuCacheLargeCount (Gem)

Number of times that a per-class lookup cache became a large object.

MILuCacheResetCount (Gem)

Number of times that a per-class lookup cache was reset.

MIPolyCacheCreateCount (Gem)

Number of times that a polymorphic send-site cache was created.

MIPolyCacheFullCount (Gem)

Number of times that a polymorphic send-site cache overflowed.

MIPolyCacheGrowCount (Gem)

Number of times that a polymorphic send-site cache was grown.

MtActiveThreads (Gem)

The number of threads actively working on a multi-threaded task in a Gem executable.

MtMaxThreads (Gem)

The maximum number of threads currently configured for a multi-threaded task in a Gem executable. Set by the method that initiated the operation.

MtPercentCpuActiveLimit (Gem)

Can be set by a session to control the maximum number of active threads working on the task. Value must be between 0 and 100. When a non-zero value is set, the controller thread attempts to keep the percentCpuActive stat below this value by limiting the number of threads working on the task.

MtThreadsLimit (Gem)

Can be set by a session to control the maximum number of active threads working on the task. Must be less than or equal to MtMaxThreads.

MultObjPagesWrittenByGem (SPC monitor)
MultObjPagesWrittenByStone (SPC monitor)

The number of multiple object pages in the cache because of a write done by a Gem or by the Stone, respectively.

NetWriteThreadSocketWrites (Stone)

The number of socket write operations attempted in stone NetWrite thread.

NetWriteThreadWakeups (Stone)

The number of wakeups from semaphore wait in stone NetWrite thread.

NewGenSizeBytes (Gem)

The number of used bytes in the new generation at the end of mark/sweep.

NewObjsCommitted (Gem)

The number of new objects committed by the most recent transaction committed by this process.

NewObjsCommittedNotLogged (Gem)

Number of newly created objects that were committed but not added to the tranlog because they are reachable from the NotTranloggedGlobals root object.

NewSymbolRequests (Gem)

The number of symbol creation requests by a session to the symbol creation Gem.

NewSymbolsCount (Gem)

The number of new symbols created by this session.

NextSleepTime (Pgsvr)

The number of milliseconds that the page server will sleep during the next sleep period. This value is adjusted to regulate the IO rate of the page server.

NotifyQueueSize (Stone)

The number of Gem session processes using notifiers.

NumCacheWarmers (Stone)

The number of cache warmer gem processes currently operating on the shared cache.

NumEphemerons (Gem)

The number of live ephemerons remaining at the end of the last mark/sweep garbage collection.

NumInLdiQueue (Stone)

The number of uncompleted requests in the stone NetLdiCall thread input list.

NumInMainInpQueue (Stone)

The number of requests from other threads in the stone main thread input list.

NumInNetReadWorkList (Stone)

The number of tasks in the NetRead thread work list, produced by NetPoll action functions and not yet processed by the NetRead thread.

NumInNetWriteQueue (Stone)

The number of uncompleted requests in the input list to the stone NetWrite thread.

NumLiveSoftRefs (Gem)

The number of instances of SoftReference in temporary object memory found during an attempt to clear SoftReferences. This counter remains at zero until temporary object space grows to at least GEM_SOFTREF_CLEANUP_PERCENT_MEM (page 368).

NumNonNilSoftRefs (Gem)

The number of instances of SoftReference in temporary object memory with non-nil, non-special values, that were found during an attempt to clear SoftReferences. This counter remains at zero until temporary object space grows to at least GEM_SOFTREF_CLEANUP_PERCENT_MEM (page 368).

NumProcsSleepingForLock (SPC monitor)

Number of processes on this shared page cache that are currently sleeping while waiting to acquire a spin lock.

NumRefsStubbedMarkSweep (Gem)

The number of references contained in copies of committed objects that were stubbed (converted to a Pom objectId) by in-memory mark/sweep.

NumRefsStubbedScavenge (Gem)

The number of in-memory references that were stubbed (converted to a Pom objectId) by in-memory scavenge.

NumSharedCounters (SPC monitor)

The number of shared counters for this shared cache.

NumSoftRefsCleared (Gem)

The number of times that the in-memory garbage collector has cleared the `value` instance variable in instances of SoftReference.

NumVotingSessions (Stone)

The number of sessions that the stone has requested to vote on possible dead objects.

ObjectMemoryGrowCount (Gem)

On AIX and HPUX only, the number of times object memory was grown or shrunk by allocating a new virtual memory region with mmap().

ObjectsRead (Gem)

The number of committed objects copied into VM memory since the start of this session.

ObjectsReadInBytes (Gem)

Total size in bytes of committed objects copied into VM memory since start of session.

ObjectsRefreshed (Gem)

The number of committed objects in VM memory that have been re-read from the page cache after transaction boundaries, since the start of this session.

ObjectTablePageReads (all)

The number of object table pages read by the process since it was last started. These page reads are actual disk reads and not reads from the shared page cache.

ObjsCommitted (Gem)

The number of objects committed by the most recent transaction committed by this process.

ObjsCommittedNotLogged (Gem)

The total number of objects (new and modified objects) reachable from the NotTranloggedGlobals root object which the session has committed.

OldestCrSession (Stone)

The session ID of a session referencing the oldest commit record. Note that more than one session may reference a commit record. A value of -1 indicates that the oldest commit record is not referenced by any session.

OldestCrSessNotInTrans (Stone)

The session ID of a session that is not in a transaction that is currently referencing the oldest commit record, which may be preventing it from being reclaimed.

OldGenSizeBytes (Gem)

The number of used bytes in the old generation at the end of mark/sweep.

OmBytesFlushed (Gem)

The total number of temporary object memory bytes flushed to Pom during commit attempts for this session.

OopNumberHighWaterMark (Stone)

The highest number of object identifiers allocated by the system.

OopsReturnedByGemsCount (Stone)

The total number of free oops returned to the stone by any gem.

OtherPageReads (all)

The number of pages read by the process that were not object table, data, or bitmap pages since the process was started. These page reads are actual disk reads and not reads from the shared page cache.

OtInternalPagesWrittenByGem (SPC monitor)
OtInternalPagesWrittenByStone (SPC monitor)

The number of object table internal pages in the cache because of a write done by a Gem or by the Stone, respectively.

OtLeafPagesWrittenByGem (SPC monitor)
OtLeafPagesWrittenByStone (SPC monitor)

The number of object table leaf pages in the cache because of a write done by a Gem or by the Stone, respectively.

PageDisposesDeferred (Stone)

The number of times a page disposal had to be deferred. This deferral can be caused by an asynchronous operation (checkpoint) being in progress on the page or by the page being attached or locked.

PageIoCount (all)

The number of page I/O (page read or page write) calls done by the process since it was last started. Each I/O call may read or write more than one page.

PageIoTimeOverallAvg (all)

The average duration of a page I/O (page read or page write) call in microseconds.

PageIoTime10SampleAvg (all)

The average duration of a page I/O (page read or page write) call in microseconds. The average is computed for the last ten I/O operations, and is updated every ten I/O operations.

PageIoTime100SampleAvg (all)

The average duration of an I/O call (page read or page write) in microseconds. The average is computed for the last 100 I/O operations, and is updated every ten I/O operations.

PageLocateCount (all)

The number of times that the process located a page. The page may have been read from disk or found in the cache.

PageMgrCompressionEnabled (Stone)

A boolean value: true if the page manager session is compressing the list of pages that it sends to remote cache page servers, false otherwise.

PageMgrPagesNotRemovedFromCachesCount (Stone)

The total number of pages that the Page Manager was unable to remove from one or more shared page caches.

PageMgrPagesPendingRemovalRetryCount (Stone)

The current number of pages that could not be removed from shared page caches by the page manager on the first attempt and are waiting to be retried.

PageMgrPagesReceivedFromStoneCount (Stone)

The total number of pages that the Page Manager session received from the Stone to remove from shared page caches.

PageMgrPageRemovalRetryCount (Stone)

Number of times the page manager session has retried removing one or more pages from the shared page caches because the first attempt to remove the pages failed.

PageMgrPagesRemovedFromCachesCount (Stone)

The total number of pages that the Page Manager has successfully removed from all shared page caches.

PageMgrPrintTimeoutThreshold (Stone)

The time threshold in seconds used by the page manager to decide if a message should be printed to its log file indicating that a remote cache page server was slow to respond.

PageMgrRemoteCachePgsvrTimeout (Stone)

Number of seconds the page manager session will wait to receive a response from a remote cache page server.

PageMgrRemoveFromCachesCount (Stone)

The total number of times that the Page Manager has attempted to remove pages from shared page caches.

PageMgrRemoveFromCachesPageCount (Stone)

The total number of pages that the Page Manager has attempted to remove from shared page caches. This statistic includes pages processed by page removal retry

operations, which occur whenever a page cannot be removed from a shared page cache on the first attempt.

PageMgrRemoveMaxPages (Stone)

The maximum number of pages the Stone will return to the page manager session in a single batch.

PageMgrRemoveMinPages (Stone)

The minimum batch size of pages that the page manager will process. The page manager will request pages to process from the Stone only if the Stone cache statistic PagesWaitingForRemovalInStoneCount exceeds this value.

PageMgrRemovePagesFromCachesPollCount (Stone)

The number of times that the Page Manager called poll() or select() to determine which cache page servers have completed removing pages from their shared caches. This statistic represents the value during the most recent page disposal operation and is not cumulative. It varies between zero (when there are no remote shared caches on the system) and the number of remote shared page caches.

PageMgrTimeWaitingForCachePgsvrs (Stone)

The total amount of real time in milliseconds that the page manager has spent waiting to receive data from remote cache page servers.

PageReads (all)

The number of pages read by the process since it was last started. These page reads are actual disk reads and not reads from the shared page cache.

PageReadsProcessingCommit (Gem)

The number of pages read while the Gem is processing its part of the commit.

PageReadsStoneCommit (Gem)

The number of pages read while the Stone is processing its part of the commit.

PageReadsWaitingForCommit (Gem)

The number of pages read while waiting for the commit token.

PageServersInCacheCount (SPC monitor)

The total number of page servers attached to the shared cache.

PagesNeedReclaimSize (Stone)

The amount of reclamation work that is pending, that is, the backlog waiting for the Reclaim GcGem to reclaim.

PagesNotFoundInCacheCount (SPC monitor)

The number of pages needing to be removed from the cache that were not found in the cache. This statistic is updated by the cache page server on remote caches, or by the page manager on the Stone's shared page cache.

PagesNotRemovedFromCacheCount (Pgsvr)

The number of pages the page server was unable to remove from the cache. Requests to remove pages come from the stone. This statistic is updated by the cache page server on remote caches, or by the page manager on the Stone's shared page cache.

PagesRemovedDirtyFromCacheCount (SPC monitor)

The number of pages successfully removed from the cache by the cache page server or the Page Manager at the Stone's request. This statistic is updated by the cache page server on remote caches, or by the page manager on the Stone's shared page cache.

PagesRemovedFromCacheCount (SPC monitor)

The total number of pages successfully removed from the cache by the cache page server or the Page Manager at the stone's request.

PagesReturnedByGemsCount (Stone)

The total number of free pages returned to the Stone by any Gem.

PagesWaitingForRemovalDeferred (Stone)

Number of deferred persistent pages waiting to be processed by the Page Manager gem.

PagesWaitingForRemovalInStoneCount (Stone)

The number of pages in the Stone that are waiting to be removed from the shared page cache by the Page Manager.

PagesWaitingForRemovalPersist (Stone)

Number of persistent pages waiting to be processed by the Page Manager gem.

PagesWaitingForRemovalTemp (Stone)

Number of temporary pages waiting to be processed by the Page Manager gem.

PageWaitQueueSize (Stone)

The size of the queue that holds sessions waiting to be allocated free pages. This queue should be empty or nearly so unless the repository is below its free space threshold.

PageWrites (all)

The number of pages written by the process since it was last started. These page writes are actual disk writes and not just writes into the shared page cache. Unless a large data load is in process, the number should be low for all processes except the Stone's AIO page server process.

PermGenSizeBytes (Gem)

The number of used bytes in the perm generation at the end of mark/sweep. Perm generation holds copies of Classes.

PersistentPagesCommittedCount (Gem)

The total number of pages made persistent (committed) by the session. This statistic is updated during commit processing.

PersistentPagesDisposed (Stone)

The number of persistent pages (pages already checkpointed) that have been disposed of while in the Stone's private cache.

PgsvrCheckpointState (Pgsvr)

The state of checkpoint processing within an AIO pgsvr:

0=not active
1=writing dirty pages

2=finished write dirty
3=in fsync

PgsvrWaitQueueSize (Stone)

The number of remote sessions waiting on page server before reclaiming session resources.

PinnedDataPagesCount (SPC monitor)

The number of pinned data pages found in the cache.

PinnedOtPagesCount (SPC monitor)

The number of pinned object table pages found in the cache.

PinnedPagesCount (all)

The number of pages the process has pinned (locked) in the shared cache. Pages may be pinned by more than one process at the same time.

PinnedPrivatePagesCount (all)

The number of pages that the process has pinned (locked) in its private page cache.

PinnedSharedCount (SPC monitor)

The number of pages pinned by more than one process.

PinnedTotalCount (SPC monitor)

The total number of pinned pages found in the cache.

PomGenScavCount (Gem)

The number of times scavenge has thrown away the oldest pom generation space.

PomGenSizeBytes (Gem)

The number of used bytes in the pom generation at the end of mark/sweep. Pom generation holds clean copies of committed objects.

PossibleDeadObjs (Stone)

The number of objects previously marked as dereferenced in the repository, but for which sessions currently in a transaction might have created a reference in their

object space. An object is not declared ("promoted to") dead until each active session verifies the absence of such references during its next commit or abort.

PostCheckpointPages (Pgsvr)

The count of pages written out by the page server during post-checkpoint processing.

PreemptedBitmapPages (Pgsvr)

The number of bitmap pages removed from the shared page cache by this page server.

PreemptedCommitRecordPages (Pgsvr)

The number of commit record pages removed from the shared page cache by this page server.

PreemptedDataPages (Pgsvr)

The number of data pages removed from the shared page cache by this page server.

PreemptedObjectTablePages (Pgsvr)

The number of object table pages removed from the shared page cache by this page server.

PreemptedOtherPages (Pgsvr)

The number of pages removed from the shared cache by this page server that were not data, object table, commit record, or bitmap pages.

PrimitiveNumber (Gem)

The primitive number currently being executed by the session, or 0 if the session is not in a primitive. The session only sets this value for long-running primitives. A non-zero value also indicates the session is immune from termination due to the STN_GEM_TIMEOUT mechanism.

Note: the PrimitiveNumber stat is set to 9999 if the session has executed System class >> disableStoneGemTimeout

ProcessId (all)

The operating system processId for the process associated with this shared page cache process slot.

ProcessName (all)

This statistic identifies the process kind (Gem, Stone, page server, or shared page cache monitor).

ProcessesWaitingForQueueLocks (SPC monitor)

Number of processes attached to the shared cache that are spinning while attempting to acquire a queue lock.

ProgressCount (Gem)

You can use this statistic to monitor the progress of certain Repository methods that may run for extended periods.

- During objectAudit, ProgressCount is set to the number of live data pages at end of the object table scan, then decremented as data pages are scanned.
- During _findPagesContainingOops:, ProgressCount is set to the total number of pages in the repository and decremented as pages are processed.
- During markForCollection, ProgressCount is incremented as live objects are marked during the marking phase, reset to zero, and incremented as possible dead objects are computed.
- During epoch garbage collection, ProgressCount is incremented as live objects within epoch are marked, then reset to zero.
- During fullBackupTo: and restoreFromBackup:, ProgressCount tracks the number of objects written to or restored from the backup file(s).
- During objectAudit, ProgressCount is incremented as the object table is scanned, then decremented as data pages are scanned.
- While reading and writing an FDC file of OOPs, ProgressCount is incremented as OOPs are read or written.

RcConflictCount (Gem)

The number of commits that resulted in an RC conflict.

RcRetryQueueSize (Stone)

The number of sessions waiting for the RcRetry object lock. (System waitForRcWriteLock:)

RebuildScavPagesForCommitCount (Gem)

The total number of times the gem rebuilt its list of scavengeable pages while processing a commit.

RecentActiveProcessCount (SPC monitor)

The number of active processes attached to the shared page cache. This statistic is computed more often than ActiveProcessCount and has decays quickly.

ReclaimCount (Stone)

The number of reclaims performed by a Reclaim GcGem process since the Stone repository monitor was last started.

ReclaimedPagesCount (Stone)

The number of pages reclaimed by a Reclaim GcGem process since the Stone repository monitor process was last started. The count indicates the number of pages that have been or will soon be placed back into the repository's pool of free pages.

RecoverFreeFrameWaitTime (Stone)

The time in seconds spent by the main recovery thread waiting for free frames in the shared page cache. Recovery performance may be improved by increasing the size of the shared page cache.

RecoverNumBufs (Stone)

The total number of logEntryBuffers allocated in the heap.

RecoverNumBufsForSessions (Stone)

The number of logEntryBuffers used to hold records for sessions not yet committed.

RecoverNumBufsInFreeList (Stone)

The number of logEntryBuffers currently in the freeList.

RecoverNumBufsInWorkQueue (Stone)

The number of logEntryBuffers currently in the workQueue.

RecoverReadThreadWaitTime (Stone)

The time in seconds that the reader thread spent waiting for the main recovery thread to catch up processing the log buffers.

RecoverReclaimOopsWaitTime (Stone)

The time in seconds spent by the main recovery thread waiting for oops to be reclaimed. Recovery performance may be improved by adding more reclaim gems.

RecoverTranlogBlockId (Stone)

The block ID of the transaction log currently being replayed during system recovery or restore.

RecoverTranlogFileId (Stone)

The file ID of the transaction log currently being replayed during system recovery or restore.

RejectedProcsCount (SPC monitor)

The total number of processes which attempted to connect to the shared page cache but were rejected because the maximum number of processes had already attached.

RemoteCachesNeedServiceCount (Stone)

The number of remote caches that require service from the Page Manager. Caches need service when they are starting or shutting down.

RemoteSharedPageCacheCount (Stone)

The total number of remote shared page caches attached to the system.

RemoteSharedPageCacheMax (Stone)

Current setting of the STN_MAX_REMOTE_CACHES configuration parameter

RepBkupRestPagesWrittenByGem (SPC monitor)
RepBkupRestPagesWrittenByStone (SPC monitor)

These statistics are obsolete; the page kind is no longer used.

ReposSizeMB (Stone)

The size of the stone's repository in MB.

Root40PagesWrittenByGem (SPC monitor)
Root40PagesWrittenByStone (SPC monitor)

The number of root 4.0 pages in the cache because of a write done by a Gem or by the Stone, respectively.

RootPagesWrittenByGem (SPC monitor)
RootPagesWrittenByStone (SPC monitor)

The number of root pages in the cache because of a write done by a Gem or by the Stone, respectively.

RunQueueSize (Stone)

The number of Gem session processes waiting for service from the Stone repository monitor.

ScavengeCount (Gem)

The number of scavenges executed by the in-memory garbage collector.

ScavengeOverflows (Gem)

The number of scavenges that overflowed in the in-memory garbage collector.

SessionId (all)

The GemStone sessionId associated with this client.

SessionPerformingBackup (Stone)

The session ID of the session that is performing a full backup; 0 if a full backup is not in progress.

SessionStat00 — SessionStat47 (Gem)

These are computed by user code to define statistics associated with a session. There are 48 session cache statistic slots available. For details, see "Global Session Statistics" on page 179.

SessionWithGcLock (Stone)

The sessionId of the session holding the GcLock. A value of 1 indicates that the lock is held by Stone recovery or restore; 0 means that the lock is not issued.

ShadowedPagesCount (Gem)

The number of data pages added to the reclaim list due to commits by this Gem. This statistic is only updated during a commit.

SigAbortsReceived (Gem)

The number of times the Stone has signaled this session to abort, that it has received and recognized.

SigAbortsSent (Gem)

The number of times the Stone has signaled this session to abort, although the session may be in a sleep or I/O wait state and not yet aware of having received the signal. (See SigAbortsReceived (Gem), above.)

SleepDuringDisposeTempPageCount (Gem)

Total number of times the Gem slept while waiting to dispose of a temporary page. Each tick of the counter represents 5 milliseconds of sleep. If you encounter excessive counts, please report them to GemStone Technical Support.

SlotsCrashedCount (SPC monitor)

The total number of slots for which the shared cache monitor has attempted recovery because a client process shutdown abnormally.

SlotsFreeCount (SPC monitor)

The number of free process slots currently available in the cache.

SlotsTotalCount (SPC monitor)

The maximum number of processes that can concurrently attach to the shared page cache.

SpinLockCount (SPC monitor)

The current setting of the SHR_SPIN_LOCK_COUNT parameter.

SpinLockFreeFrameSleepCount (SPC monitor)

The number of times the process was forced to sleep on a semaphore while attempting to acquire the free frame list spin lock. Available only for shared page cache monitor slot.

SpinLockFreePceSleepCount (SPC monitor)

The number of times the process was forced to sleep on a semaphore while attempting to acquire the free page cache entry spin lock. Available only for shared page cache monitor slot.

SpinLockHashTableSleepCount (SPC monitor)

The number of times the process was forced to sleep on a semaphore while attempting to acquire a hash table spin lock. Available only for shared page cache monitor slot.

SpinLockNewSymSleepCount (SPC monitor)

The number of times a process was forced to sleep on a semaphore waiting for the NewSymbols spinLock.

SpinLockOtherSleepCount (SPC monitor)

The number of times the process was forced to sleep on a semaphore while attempting to acquire either the AllSymbols or shared counter spin lock. Available only for shared page cache monitor slot.

SpinLockPageFrameSleepCount (SPC monitor)

The number of times the process was forced to sleep on a semaphore while attempting to acquire a page frame spin lock. Available only for shared page cache monitor slot.

StatsThreadTimeRunningMs (SPC monitor)

Approximate real time in milliseconds that the statistics thread in the shared page cache monitor has spent doing work.

StnAioCompletionFailures (Stone)

The number of aio_write() operations for which either aio_error() or aio_return() reported that the AIO failed.

StnAioFsyncFailures (Stone)

The number of fsync() operations in AioWait thread which failed.

StnAioMainTimeInAioWrite (Stone)

Total real time in milliseconds that the main thread spent executing the aio_write() call. aio_write() is used to initiate asynchronous writes to the tranlog and should not block.

StnAioNumWriteThreads (Stone)

Value of the STN_NUM_AIO_WRITE_THREADS Stone configuration parameter, which determines how many threads are dedicating to writing to the tranlog.

StnAioWaitsForWork (Stone)

The number of times the Stone's AIO wait thread waited on semaphore for more work.

StnAioWaitTotalTime (Stone)

Approximate total real time in milliseconds that the AIO wait thread spent waiting for asynchronous I/O requests to complete.

StnAioWriteFailures (Stone)

The number of aio_write() calls by the Stone main thread which failed with other than EAGAIN. Stat should normally be zero.

StnAioWriteQueueHighWaterSize (Stone)

High water mark of the StnAioWriteQueueSize statistic.

StnAioWriteQueueSize (Stone)

Size of the tranlog write request queue.

StnAioWritesQueuedCount (Stone)

Total number of tranlog write requests that have been queued.

StnAioWriteThreadsIdle (Stone)

Number of tranlog write threads in the Stone process that are currently idle.

StnCrBacklogThreshold (Stone)

Current setting of the STN_CR_BACKLOG_THRESHOLD Stone configuration parameter.

StnGetLocksCount (Stone)

The total number of times the Stone retrieved the lock set and passed it to a remote gem.

StnLoopCount (Stone)

The total number of times the Stone has executed its service loop. If this number remains unchanged for a significant period (for example, ten seconds or so), the Stone has hung.

StnLoopHibernateCount (Stone)

The total number of times the Stone went to sleep waiting for a network event to occur. This state occurs when StnLoopState is set to 17.

StnLoopNoWorkThreshold (Stone)

Current setting of the STN_LOOP_NO_WORK_THRESHOLD stone configuration parameter.

StnLoopsNoWork (Stone)

Number of times the stone has executed its main service loop and found no work to perform. This counter is reset to zero each time the stone performs work.

StnLoopsSinceSleep (Stone)

Number of times the stone has executed its main service loop since sleeping. This counter is reset to zero each time the stone sleeps while waiting for work.

StnLoopState (Stone)

An integer that identifies where, in the Stone control loop, the Stone process is currently executing. For a meaningful statistic, set your sample rate to faster than a second. For state definitions, consult GemStone technical support.

StnMainWaitsForFreeAio (Stone)

The number of times the stone main thread waited for free AIO buffers to become available.

StnTranQToRunQThreshold (Stone)

Current setting of the STN_TRAN_Q_TO_RUN_Q_THRESHOLD stone configuration parameter.

StoneCommitState (Stone)

This internal statistic is used to determine the state of the Stone's commit processing.

SymbolCreationQueueSize (Stone)

The number of sessions waiting for a Symbol creation request to be processed.

TargetFreeFrameCount (SPC monitor)

The minimum number of unused page frames the free frame page server(s) will attempt to keep in the cache. The free frame count can still fall below this value if the cache contains mostly dirty pages, which free frame page servers cannot preempt.

TargetPercentDirty (Pgsvr)

The percent of dirty pages that AIO page servers try to maintain in the shared cache. If the dirty pages are below this target then the I/O rate will be limited on the next scan. If the dirty pages are above this target then the I/O rate will be set to AioRateMax.

TempObjSpacePercentUsed (Gem)

The approximate percentage of the total reserved temporary object memory for this session which is in use. Sessions are likely to encounter an out of memory error if this value approaches or exceeds 100%. This statistic is only updated at the end of a mark/sweep operation.

TempPagesDisposed (Stone)

The number of temporary pages (pages allocated since the last checkpoint) that have been disposed.

TimeInCommitRecordDisposal (Stone)

The total amount of real time in milliseconds that the Stone has spent disposing commit records. Commit records disposed during repository startup are not included in this statistic.

TimeInFramesFromFindFree (all)

The cumulative number of milliseconds that the Gem or Stone has spent scanning the shared page cache for a free frame since the session (for Gems) or Stone started.

TimeInMarkSweep (Gem)

The real time in milliseconds spent in in-memory garbage collector mark/sweeps.

TimeInPgsvrNetReads (Stone, Gem)

The cumulative amount of real time in milliseconds that a session or Stone has spent reading network data from a page server.

TimeInPgsvrNetWrites (Stone, Gem)

The cumulative amount of real time in milliseconds that a session or Stone has spent writing network data to a page server.

TimeInScavenges (Gem)

The real time in milliseconds spent in in-memory garbage collector scavenges.

TimeInStnGetLocks (Stone)

The total time spent by the Stone retrieving the lock set and passing it to a remote gem.

TimeInStonePageDisposal (Stone)

The elapsed real time in milliseconds the Stone spent performing page disposal operations.

TimeInUpdateUnionsCommit (Gem)

The total real time the gem spent updating its unions while waiting for the commit token.

TimeInWaitsForOtherReaders

The real number of milliseconds the process spent waiting for the read of another process to complete.

TimeLastEpochGc (Stone)

Time at which the Admin GcGem finished computing dead objects for the last epoch. Time to end of the currently open epoch is computed from here.

TimePerformingCommit (Stone)

The number of milliseconds of real time the Stone has spent performing commit processing, not including time taken by asynchronous writes to the transaction log.

TimePerformingReadIo (Pgsvr)

The total amount of real time in milliseconds the page server has spent reading pages from disk.

TimePerformingReadRequests (Pgsvr)

The total amount of real time in milliseconds the page server has spent performing read requests for its client. This statistic includes time reading pages from disk and time searching the shared page cache for pages.

TimeProcessingCommit (Gem)

The cumulative amount of time in milliseconds that the Gem session process has spent doing the processing for commits while it has the commit token.

TimerThreadWakeups (Stone)

The number of wakeups from sleep in stone Timer thread.

TimeSleepingMs (Pgsvr)

Total real time the process has spent sleeping, in milliseconds.

TimeStoneCommit (Gem)

The cumulative amount of time in milliseconds that the Gem session process has waited for the Stone repository monitor to complete commits by this session.

TimeWaitingForCommit (Gem)

The cumulative amount of time in milliseconds that the Gem session process has spent waiting for its turn to commit, that is, the time waiting for the commit token and the Stone's processing time for serialization.

TimeWaitingForIo (all)

The total real time in milliseconds that the process has spent waiting for I/O calls that read or write pages to complete. Only page I/O is included in this statistic. Other types of I/O (such as transaction log writes by the Stone process) are not included.

TimeWaitingForStone (Gem)

The total time the Gem spent waiting for a response from the Stone.

TimeWaitingForSymbols (Gem)

Cumulative elapsed time in milliseconds waiting for symbol creation requests to be processed.

TotalAborts (Stone)

The number of abort operations performed system-wide since the Stone was started.

TotalBmPageReads (SPC Monitor)

Total number of bitmap pages read by all processes currently attached to the shared page cache.

TotalCommits (Stone)

The total number of commits (excluding read-only commits) performed by all processes since the Stone repository monitor was last started.

TotalDataPageReads (SPC Monitor)

Total number of data pages read by all processes currently attached to the shared page cache.

TotalFramesAddedToFreeList (SPC Monitor)

Total number of frames added to the free list by all processes currently attached to the shared page cache.

TotalFramesFromFindFree (SPC Monitor)

Total number of frames found by scanning the cache for all processes currently attached to the shared page cache.

TotalFramesFromFreeList (SPC Monitor)

Total number of frames taken from the free list by all processes currently attached to the shared page cache.

TotalGemFatalErrors (Stone)

The total number of fatal errors reported to the stone by all gems.

TotalLocalPageCacheHits (SPC Monitor)

Total number of local page cache hits for all processes currently attached to the shared page cache.

TotalLocalPageCacheMisses (SPC Monitor)

Total number of local page cache misses for all processes currently attached to the shared page cache.

TotalLostOtsSent (Stone)

The total number of SigLostOtRoot signals sent by the stone.

TotalMiscPageReads (SPC Monitor)

Total number of miscellaneous pages read by all processes currently attached to the shared page cache.

TotalNewObjsCommitted (Stone)

The total number of new objects committed by all Gems.

TotalOtPageReads (SPC Monitor)

Total number of object table pages read by all processes currently attached to the shared page cache.

TotalPageReads (SPC Monitor)

Total number of pages read by all processes currently attached to the shared page cache.

TotalPageWrites (SPC Monitor)

Total number of pages written by all processes currently attached to the shared page cache.

TotalPcesAddedToFreeList (SPC Monitor)

Total number of PCEs (Page Cache Entries) added to the free list by all processes currently attached to the shared page cache.

TotalPcesInFreePceCaches (SPC Monitor)

Total number of free PCEs (Page Cache Entries) present in the free PCE caches of all processes currently attached to the shared page cache.

TotalPcesRemovedFromFreeList (SPC Monitor)

Total number of PCEs (Page Cache Entries) removed from the free list by all processes currently attached to the shared page cache.

TotalProcsInCacheCount (SPC monitor)

he total number of processes currently connected to the cache, including crashed processes that have not yet had their cache slot recovered.

TotalSessionsCount (Stone)

The total number of sessions currently logged in to the system.

TotalSessionsStopped (Stone)

The total number of stop session requests initiated by a gem or by the stone.

TotalSessionsTerminated (Stone)

The total number of sessions forcibly logged off by stone.

TotalSigAbortsSent (Stone)

The total number of SigAborts sent by the stone.

TotalWaitsForOtherReader (SPC Monitor)

Total number of PageRead operations avoided by all processes currently attached to the shared page cache.

TotEphemeronsFired (Gem)

Total number of ephemerons whose key has been garbage collected. Only updated at the end of a mark/sweep GC.

TrackedSetSize (Gem)

The number of objects in the Tracked Objects Set, as defined by the GCI.

TranlogFileId (Stone)

The file id of the transaction log to which the most recent tranlog entry was written.

TranlogKBytesWritten (Gem)

Total number of KB written to the tranlog by stone on behalf of this session.

TranlogRecordId (Stone)

The record id of the most recent transaction log entry to be written.

TranlogRecordKind (Gem)

The kind of tranlog record last written to the tranlog by stone on behalf of this session.

TranlogRecordsWritten (Gem)

Total number of physical tranlog records written to the tranlog by stone by this session.

TranlogsFull (Stone)

A flag indicating if all configured tranlog partitions or directories are full; 1 means full.

TransactionLevel (Gem)

This statistic describes the state of the Gem: 1 (in a transaction), 0 (outside a transaction), or –1 (a transactionless state in which the Stone will never signal the Gem to abort).

TteCrPageFreeCount (Stone)

Total number of free commit record page entries in the stone's internal commit record cache.

TteCrPageFreePoolSize (Stone)

Total number of commit record page entries allocated in the stone's internal commit record cache.

TteFreeCount (Stone)

Total number of free TTEs (transaction table entries) in stone.

TteFreePoolSize (Stone)

Total number of TTEs (transaction table entries) allocated by stone.

UpdateUnionsCommitCount (Gem)

The total number of times the gem updated its unions while waiting for the commit token. This count will be at least one for every commit.

VoteNotDead (Gem)

The number of objects that the Gem process removed from the possible dead set the last time that it voted on the possible dead.

VoteNotDeadObjs (Stone)

The number of objects that the Gem process removed from the possibleDead set the last time that it voted on the possible dead.

WaitingForSessionToVote (Stone)

The sessionId of a session which the system is waiting for to complete the voting on possible dead objects. A zero value indicates that it is not waiting.

WaitsForOtherReader (all)

The number of PageRead operations avoided by waiting for read already in progress by another process.

WorkingSetSize (Gem)

The number of objects in memory that have an object Id assigned to them. Approximately the number of committed objects that have been faulted in plus the number that have been created.

Appendix

I Object State Change Tracking

I.1 Overview

GemStone transaction logs (tranlogs), which provide a way to recover all changes made to the repository in the case of repository crashes, or allow warm standbys to apply changes made to a primary repository, include detailed records of all changes in a encoded and compressed form. Converting the tranlog information to human-readable form, and analyzing this output, provides invaluable information for debugging and testing. It allows us to determine exactly what changes have been made to any of the objects in the repository over time. The tools used to perform this have been used internally to GemStone for many years, and have been provided to customers on occasion when needed to analyze specific application problems.

The ability to track changes to objects may be useful for customers who need to identify details on changes that have been made to application data objects. For this reason, we are making these scripts available as part of the GemStone product. Additional information has been added to the tranlogs to allow tracking of the specific user or machine that originated the object changes.

The scripts used to perform tranlog analysis are located in the `$GEMSTONE/bin/` directory and are named:

`printlogs` — to output the complete contents (optionally filtered) of selected tranlogs in human-readable form.

`searchlogs` — to search all tranlogs in a directory for selected OOPs (Object Oriented Pointers, or Object Ids), and output the matching entries (optionally filtered) in human readable form.

A GemStone repository performs many automatic operations, including things like garbage collection and checkpoints, that are transparent to end users. The tranlogs, of course, must contain records of any changes made by these operations. Complete details on everything that tranlogs may contain is beyond the scope of this documentation. The information provided here is intended to allow the use of the tranlog analysis scripts to locate and identify the details of changes to application objects.

Object oriented design's principle of encapsulation allows you to hide internal object complexity. However, to understand the data recorded in the tranlogs, you must have a detailed understanding of the actual structure of the objects. This includes both your own application classes, and the classes that are part of GemStone Smalltalk.

Also, note that since the tranlog analysis scripts are general purpose, used for a wide variety of purposes in which a detailed record of internal repository operation is required, the scripts may output much more information than is necessary for tracking object state changes. You may need to ignore this extraneous information as you perform your analysis.

I.2 Tranlog Analysis Scripts

Script Prerequisites

The environment variable `$GEMSTONE` must be set, and the `$GEMSTONE/bin/` directory must be in your executable path.

The scripts perform the analysis on tranlogs that are in the current working directory. If you are using raw partitions for your tranlogs, locate disks on the file system with adequate space for both the tranlogs themselves, and the script output files, which may be larger than the original tranlogs. Use `copydbf` to copy the tranlogs from the raw partition to the file system. For more information on the `copydbf` utility, see Appendix B, "GemStone Utility Commands.".

Output

Output from the scripts goes directly to `stdout`. To preserve the output and allow it to be used in later steps of analysis, redirect this; for example:

```
$> printlogs tranlog1.dbf > tranlog1.out
```

Note that these scripts can produce very large amounts of output, so make sure that you have adequate disk space.

In some cases the resulting files may be too large for unix text editors such as `vi` or `emacs` to open. You may find it necessary to use the unix `split` utility to break up very large output files into more manageable chunks.

Tranlog Assumptions

By default, the tranlogs are assumed to be named using the GemStone convention `tranlog`*NNN*`.dbf`. If you are using a different naming convention, you can override this by setting the environment variable `$GS_TRANLOG_PREFIX` to the prefix you are using.

The scripts use the creation date of the tranlog file to determine the order in which the tranlogs are analyzed. If you copy or manipulate the tranlogs in a way that changes the creation date, this may cause the analysis to be done out of order. The output will warn of this with the message:

```
*** Warning:  scanning tranlogs out of order
```

Filter Criteria

The scripts both allow you to filter the results, to locate entries that are related to a particular UserId, GemHost, or ClientIP.

UserId – The userId (user name) of the UserProfile associated with this session: DataCurator, SystemUser, etc. The filter keyword is `user`.

GemHost – The name or IP address of the host machine running the gem process. For a linked session, which links the gem into the client, this is the same machine as the client.

If the gem is running on the same machine as the stone, use the name of the host machine. Otherwise, if the gem is on a different machine than the stone, use the IP address of the remote machine.

The filter keyword is `host`.

ClientIP – The IP address of the host machine running the client process.

For an RPC session, this is the machine running the client application. Clients may be topaz -r, GemBuilder for Smalltalk, or GemBuilder for C applications. For a linked session, this is the machine running the linked client/gem (the same machine as the GemHost). However, the ClientIP is always the IP address, even if it is on the same machine as the stone.

The filter keyword is `client`.

Effective UNIX user ID – The integer that is the effective UNIX user id of the gem process.

The filter keyword is `euid`.

UNIX user ID – The integer that is the real UNIX user ID of the gem process.

The filter keyword is `ruid`.

effective UNIX user name **–** The effective UNIX user name of the gem process.

The filter keyword is `euidstr`.

UNIX user name – The real UNIX user name running the gem process.

The filter keyword is `ruidstr`.

process ID – The integer process id (PID) of the gem process.

The filter keyword is `gempid`.

session ID – The integer session id of the session within GemStone.

The filter keyword is `sessionid`.

WARNING

Information about UserId, GemHost, and ClientIP are derived from a ***Login*** *tranlog entry created when a session first logs in. This entry associates the UserId/GemHost/ClientIP with a particular sessionID, which is then used as a key for subsequent tranlog entries. If you start analysis from a later tranlog which does not include this* ***Login*** *entry, these fields will be left blank for that session, and printouts/searches*

using filters based on these fields will not locate any results. Likewise, scanning through tranlogs out of order may result in the wrong ***Login*** *entry being associated with a given sessionID. This would set UserId/GemHost/ ClientIP incorrectly for that particular session, and produce incorrect results when filtering.*

printlogs

This script prints out the contents of one or more tranlogs in the current working directory in a human-readable form.

Warning

This script produce a very large amount of output, which (unfiltered) will exceed the size of original tranlog/s, and depending on the contents may be twice as large as the original tranlogs. Consider disk space, the use of filters, and restricting the set of tranlogs before running this script. Use caution in including the `full` *keyword.*

Usage:

```
printlogs [<filters>] [full] [all] [<tlogA> ... <tlogZ>]
```

If <*filters*> are specified, only print out the tranlog entries that match the specified criteria. Filters may be combined. Possible filters are:

`user` <*userId*> – Filter by the userId (the user name) of the GemStone UserProfile
`host` <*hostnameOrIP*> – Filter by gem/topaz process host or IP address
`client` <*N.N.N.N*> – Filter by client IP Address
`euid` <*integer*> – Filter by gem's effective UNIX user ID
`ruid` <*integer*> – Filter by gem's real UNIX user ID
`euidstr` <*string*> – Filter by gem's effective UNIX user name
`ruidstr` <*string*> – Filter by gem's real UNIX user name
`gempid` <*integer*> – Filter by gem's process ID
`sessionid` <*integer*> – Filter by gem's session ID

`full`— Produce more detailed information. WARNING: this produces much larger output results.

`all` —Print out the contents of all tranlogs in the current working directory.

Examples

To print out the entire contents of all tranlogs in this working directory:

```
printlogs all
```

To print out all entries in a selected number of tranlogs (note that tranlogs in the sequence must be contiguous):

```
printlogs tranlog5.dbf tranlog6.dbf tranlog7.dbf
```

To print out all tranlog entries for the user DataCurator in any tranlog:

```
printlogs user DataCurator all
```

To print out detailed information for all entries in tranlog5.dbf for the user DataCurator:

```
printlogs full user DataCurator tranlog5.dbf
```

searchlogs

This script prints out tranlog entries associated with particular OOP values, according to the arguments in the command line. All tranlogs in the current working directory are scanned.

Usage:

```
searchlogs [<filters>] <oopA> ... <oopB>]
```

If *<filters>* are specified, only print out the tranlog entries that match the specified criteria. Filters may be combined. Possible filters are:

`user` **<***userId***>** — Filter by the userId (the user name) of the GemStone UserProfile
`host` **<***hostnameOrIP***>** — Filter by gem/topaz process host or IP address
`client` **<***N.N.N.N***>** — Filter by client IP Address
`euid` *<integer>* — Filter by gem's effective UNIX user ID
`ruid` *<integer>* — Filter by gem's real UNIX user ID
`euidstr` *<string>* — Filter by gem's effective UNIX user name
`ruidstr` *<string>* — Filter by gem's real UNIX user name
`gempid` *<integer>* — Filter by gem's process ID
`sessionid` *<integer>* — Filter by gem's session ID

When using more than one filter, you must list the filters in the listed order.

Examples

To print out all entries involving OOP 1234 and OOP 5678:

```
searchlogs 1234 5678
```

To print out all entries involving OOP 1234 performed by DataCurator:

```
searchlogs user DataCurator 1234
```

To print out all entries involving OOP 1234 and OOP 5678 performed by DataCurator while logged in from client machine 10.20.30.40:

```
searchlogs user DataCurator client 10.20.30.40 1234 5678
```

I.3 Tranlog Structure

In order to make sense of the output from the scripts, you need a basic understanding of how tranlogs are structured.

GemStone transaction logs consist of a sequence of **tranlog records**. Tranlog records are written on **physical pages** of 512 bytes (note that this is different from the larger page size used for extents); a tranlog record may extend over more than one page, but its size is always a multiple of 512 bytes.

Each tranlog record contains one or more **tranlog entries** (sometimes referred to internally as logical records). The tranlog entries contain the information of interest - the actual changes to objects in the repository (and any other repository operations).

Output from the scripts will include header information for the tranlog record (see Example I.1), followed by data from each of the tranlog entries held by that tranlog record.

Example I.1 Tranlog Record Header Output

```
Dump of record 11 hdr.pageId 42949672962 , Kind=(16)Tran Log Record
 thisRecordId:(file:2 rec:11) previousRecordId:(file:2 rec:10)
 recordSize: 1024 numLogicalRecs: 2 fileCreationTime: 1297200655
```

Tranlog records are identified by the pageId that they begin on. Since a tranlog record may extend over multiple pages, there may be a gap in the sequence of record ids in the output. For example, the first tranlog record in Example I.2, record 7, has a recordSize of 1536 (three 512-byte physical pages), and so the next tranlog record will be 10.

Example I.2 Gap in Tranlog Record Sequence

```
Dump of record 7 hdr.pageId 25769803778 , Kind=(16)Tran Log Record
 thisRecordId:(file:2 rec:7) previousRecordId:(file:2 rec:6)
 recordSize: 1536 numLogicalRecs: 2 fileCreationTime: 1297200655

2.7.0  BeginData session: 4 beginId:(100979 1)
  numObjs:3  pad1:0 pad2:0
 2195969  240385  20423937

2.7.1  Commit session: 4 beginId:(100979 1)  crPage: 1039 commitSeq:
0.158  timeWritten: 02/08/11 13:31:52 PST
  seqType:normal beginLogRecord:(file:2 rec:7)
---------------------------------------
Dump of record 10 hdr.pageId 30064771074 , Kind=(16)Tran Log Record
 thisRecordId:(file:2 rec:10) previousRecordId:(file:2 rec:7)
 recordSize: 512 numLogicalRecs: 3 fileCreationTime: 1297200655
```

Tranlog Entries

Each tranlog entry contains a unique identifying code, a descriptive entry type, and information on the session that originated the tranlog entry.

The identifying code consists of three numbers in the form:

```
AAA.BBB.CCC
```

where:

`AAA` – the fileId of the tranlog holding the entry
`BBB` – the pageId for the tranlog record holding the entry
`CCC` – the entryId: the number of the entry within the tranlog record

Each tranlog entry is of a particular type, according to the event that it represents and the information it contains. Types are indicated by short descriptive terms such as **Login**, **Abort**, **Commit**, and **Data**. There are a large number of entry types, most of which are not important for tracking object state changes and can be ignored (for example, a **Checkpoint** entry. The ones that are important are documented below.

Each tranlog entry is associated with an Integer sessionID. SessionsIDs are unique to a specific session at any point in time. However, when a session logs out, the

sessionID becomes available again for reuse by a new session logging in, so over a period of time, a sessionId may be used by a number of different sessions.

A sessionID of zero is used for tranlog entries generated by the stone.

If the entry was not originated from the stone (that is, the tranlog entries sessionID is not 0), the tranlog entry header includes more information about the session: the UserId, the GemHost, and the ClientIP address. These are described in more detail under "Filter Criteria" on page 527.

Example I.3 Example Tranlog Entry

```
2.3.4  Login session: 4 beginId:(100973 1) userProfileOop: 208385
timeWritten: 02/08/11 13:30:56 PST userId: SymbolUser gemhost: myhost
clientIP: 10.20.30.40 processId: 12663 rUid: 531 eUid: 531 realU:
gsadmin effU: gsadmin

2.3.5  StartSymbolGem session: 4 beginId:(100973 5)
```

Example I.3 shows that is in the tranlog with fileId 2 (by the default naming convention, `tranlog2.dbf`), it is in the third physical page and in tranlog record 3, and these are the fourth and fifth tranlog entries in tranlog record 3.

These entries are of the tranlog entry types `Login` and `StartReclaimGcGem`. The session is logged in as the user named `SymbolUser`; the gem session is executing on the same machine as the stone, a machine named `myhost`; the client is executing on a machine with the IP address `10.20.30.40`; and the OS process is owned by the OS userid `gsadmin`. (beginId contains transaction tracking information that you can ignore).

Other information will follow this basic data, depending on the type of entry.

Tranlog Entry Types

There are a large number of tranlog entry types. Most of these are not relevant to tracking object state history - they record internal operations of the system, such as garbage collection or checkpoints. These tranlog entry types are not documented, although their functions can often be deduced by their names.

Below are the entries important for tracking object state history:

Login

A new session is logging into GemStone. As mentioned earlier, this entry logs the UserId, GemHost, ClientIP and other data for this particular sessionID. If you start analysis on a tranlog that follows this event, these fields will be left blank for that session.

For example, if session 7 logs in while tranlog4 is in effect, and logs out when tranlog5 is in effect, and you begin analysis on tranlog5, entries for this session will not contain any session detail information. If sessionID 7 is reused by a new session logging in later during tranlog5, that login will be recorded in tranlog5, and subsequent entries for this new session will be displayed properly.

AbortLogout

This entry is written when a session logs out or the Stone detects that the session has been killed. This entry is not flushed to the transaction log until a commit occurs.

BeginData
Data
BeginStoreData
StoreData

GemStone uses the above four entry types for recording new or changed object data. The basic entry information is followed by additional information about the changes, including the OOP values of the changed objects. Using the optional "full" flag in the `printlogs` script will cause the output to include additional information on the exact changes made.

Commit

All the changes recorded in earlier **BeginData**, **Data**, **BeginStoreData**, or **StoreData** entries are now officially committed to the repository.

Abort

Any changes recorded earlier in this transaction are discarded.

BreakSerialization

This entry indicates that a transaction conflict occurred during an attempt to commit. Any changes recorded earlier in this transaction are not yet permanent. This event is usually followed by an **Abort** entry, although it's possible that the

next event seen for this sessionID is a **Login**, if the original session logged out and a new session reuses the sessionID.

Very Large Objects

GemStone is designed so that all objects will fit on a single page of 16384 bytes. This means that a byte object can be no larger than about 16000 bytes (since page header information uses some space), and pointer objects can only have about 4000 elements. GemStone internally represents objects larger than this as a tree structure, where the root node object references 2 or more leaf node objects, which then reference the actual elements of the collection object. Extremely large objects, such as large collections, may have internal branch nodes, if the number of leaf objects needed exceeds 4000.

This internal structure is usually transparent to the user. So, for example, you may create and manipulate an Array containing 20,000 elements as if it was a single large object, while the actual representation is a root object that references five leaf objects, each containing a 4000-element chunk of the array. While this makes application development with GemStone much simpler, the entries in the tranlogs reflect the actual implementation; you will need to be aware of this to understand tranlog output relating to collections larger than ~4000 object references or ~16000 bytes. Adding an element to the large Array, for example, may mean the tranlog entry includes a change to an instance of LargeObjectNode (the leaf node object), rather than a change to the Array itself.

Full vs. Normal Mode

When using the printlogs script, you can optionally specify "full" mode to get more details on the changes made to objects in the repository. But this will greatly increase the size of the resulting log files. For example, using normal mode on our test tranlogs generated a log file that contained an entry that looked like this:

Example I.4 Tranlog entry, normal mode

```
2.11.0  BeginData session: 5 beginId:(100978 1)
  numObjs:3 pad1:0 pad2:0
 25567233  8532993 25880577
```

which tells you that three objects were created or modified during this event, with oops 25567233, 8532993, and 25880577.

Using "full" mode will produce a much more detailed listing for this event:

Example I.5 Tranlog entry, full mode

```
2.11.0  BeginData session: 5 beginId:(100978 1)
  numObjs:3 pad1:0 pad2:0

objId 25567233 class 114177 segId:9 len 14 gcSz 14 psize 136 bits 0x1
page 1152
Oop values:  50(sI 6) 8532993  5042177  25881601  17979137  26620161
20272897  25801985
8:  20322817  25559553  20324097  25559297  20423937  25869569

objId 8532993 class 111361 segId:9 len 14 gcSz 14 psize 136 bits 0x1
page 1152
Oop values:  162(sI 20) 122(sI 15) 20002(sI 2500) 42(sI 5) 20
18224129  20  25567233
8:  20  25566977  20  25801729  20  25566721

objId 25880577 isNewObj=1  class 73985 segId:9 len 23 gcSz 23 psize
208 bits 0x241 page 1152
Oop values:  18(sI 2) 18(sI 2) 20  2(sI 0) 25870081  25871105  20  20
8:  20  20  20  20  20  20  20  20
16:  20  20  20  20  20  20  20
```

Note that there is now a description of each individual created or modified object, containing these fields:

`objId`: The OOP of this object

`isNewObj`: A Boolean indicating if the object is a newly created object.

`class`: The OOP of the class of this object

`segId`: The id of the GsObjectSecurityPolicy (formerly Segment) associated with this object

`len`: The logical size of this object (in bytes for a byte object, OOPs for an OOP object)

`gcSz`: The logical size of this object (in bytes for a byte object, OOPs for an OOP object)

`psize`: The physical size of this object on disk in bytes (including 20 bytes of object header information)

`bits`: The format bits for this object (internal GS use)

`page`: The extent page that this object is written on

`Bytes`: The actual bytes that make up this object (if a byte object)
[or]
`Oop values`: The actual OOPs that make up this object (if an OOP object)

If the Oop values contain more than eight elements, they are broken into lines of eight items, each of which is prefixed by a counter. For example, an Array of 30 items might look like this:

```
objId 39056641 isNewObj=1  class 66817 segId:9 len 30 gcSz
30 psize 264 bits 0x201 page 1005
Oop values:  39056385  39056129  39055873  39055617
39055361  39055105  39054849  39054593
8:  39054337  39054081  39053825  39053569  39053313
39053057  39052801  39052545
16:  39052289  39052033  39051777  39051521  39051265
39051009  39050753  39050497
24:  39050241  39049985  39049729  39049473  39049217
39048961
```

Bytes are broken up similarly, but the sections are 60 bytes rather than 8. For example, the source code string for the `name:` method might look like this:

```
objId 38711809 isNewObj=1  class 74753 segId:9 len 91 gcSz 0
psize 120 bits 0x280 page 1097
Bytes:  name: newValue

   "Modify the value of the instance variabl
     61: e 'name'."
   name := newValue
```

I.4 Example of Tranlog Analysis

For this example, we created a simple database containing some Employee information and performed some simple operations creating and modifying these Employee objects.

The structure of classes associated with the Employee data is as follows:

Employee:

name - a Name object
age - a SmallInteger
address - an Address object

Name:

last - a String object
first - a String object
middle - a String object

Address:

addr1 - a String object
addr2 - a String object
city - a String object
state - a String object
zip - a SmallInteger

After having User1 create five Employee objects (with associated Name and Address objects), we then had User1 and User2 make some minor changes to one of the Employees:

- User2 incremented the age after a birthday.
- User1 changed the address after a move.

When completed, we had two tranlogs, tranlog2.dbf and tranlog3.dbf.

Tracking Changes to an Employee

Let's say we want to examine the change history of a particular Employee. Using the method #asOop, we find the OOP of the Employee object of interest is 38808321.

```
topaz 1> printit
| myEmployee |
myEmployee := <code to locate employee object>.
myEmployee asOop.
%
38808321
```

The data composing an Employee is contained in subobjects (such as address), as well as directly (such as age). So, we will also need to track changes to these subobjects. Again using #asOop, we find that the OOP of the Name object associated with this Employee is 155073, and that the OOP for the Address object is 155069.

```
myEmployee name asOop
38808577
myEmployee address asOop
38808833
```

You can use #asOop on any persistent object in the repository. For example,

```
73 asOop
586
nil asOop
20
```

We can now search our tranlogs for any events involving these objects. Using the command:

```
$> searchlogs 38808321 38808577 38808833
```

results in the following output:

Example I.6 seachlogs output for Employee

```
Searching for oops: 38808321 38808577 38808833
Searching tranlogs:

  tranlog2.dbf
  tranlog3.dbf
```

... header material omitted ...

```
2.118.0  BeginData session: 5 beginId:(101033 0)  newobj  38808321
cls 39017217 onPage 1081,  newobj  38808577 cls 38726401 onPage 1081,
newobj  38808833 cls 38715649 onPage 1081

2.123.0  Commit session: 5 beginId:(101033 0)  timeWritten: 02/08/11
14:43:42 PST

2.215.0  BeginData session: 6 beginId:(101664 1)  object  38808321
cls 39017217 onPage 871,

2.215.1  Commit session: 6 beginId:(101664 1)  timeWritten: 02/08/11
16:14:48 PST

3.22.0  BeginData session: 8 beginId:(101779 1)  object 38808833 cls
38715905 onPage 900,

3.22.1  Commit session: 8 beginId:(101779 1)  timeWritten: 02/08/11
16:32:58 PST
```

From this output, we can see that in tranlog entry 2.118.0, Session 5 made changes to all three objects (in this case, when the Employee and associated subobjects were first created). In entry 2.215.0, session 6 made a change to the Employee, and then later in entry 3.22.0, session 8 made a change to `38808833`, the Address object.

Note that the **BeginData** entries are each followed by a **Commit**. You should always confirm that a **BeginData**/**Data**/**BeginStoreData**/**StoreData** of interest is followed by a **Commit**. If it doesn't then the reported event was not made persistent in the repository.

Changed vs. new objects

In the above example, while the field of an Address object changed, the Address object itself was the same (had the same OOP). Depending on how the Smalltalk application is written, this may not always be the case. If application that was initiating these changes created a new Address object, and assigned the Employee's address instance variable to this new object, then the Employee object would reference a new OOP, rather than OOP 38808833. This would make the analysis somewhat different. For example, in the initial stage of the analysis when you look up the OOP of the Address object in your application, you would find the new OOP rather than the original OOP. Looking back in time, you would see when this Address object was created and assigned to the Employee instance.

Details of Changes to an Employee

Having used the `searchlogs` script to get a general idea of which tranlogs are of interest, you can now use the `printlogs` script to get more details.

For example, you might want more details on the creation of Employee object 38808321 and its associated subobjects 38808577 and 38808833 in entry `2.118.0`. The "2" in "`2.118.0`" indicates that `tranlog2.dbf` is the tranlog of interest. The command:

```
$> printlogs tranlog2.dbf
```

will generate a condensed listing of all events in `tranlog2.dbf`. By searching the resulting file for the entry number 2.118.0 you can find the relevant entry:

Example I.7 Employee modification

```
3.61.0  BeginData session: 2 user: User1 gemhost: merlin clientIP:
10.20.30.40 beginId:(134 0)
  clusterId: 1,  extentId: 0  numObjs:35

145021  155041  155045  155049  155053  155069  155073  155077
155089
155093  155097  155101  155121  155273  155285  155317  156589
156597  156609  156613  156661  156689  156729  156733  156749
156825  156829  156833  156837  156857  156861  156885  179629
180045  180653
```

This shows the oops of *all* objects created during this event. If you want to see more details on the actual changes made, use the "full" argument in the `printlogs` command:

```
$> printlogs full tranlog3.dbf
```

This will produce a more detailed listing of all events. For tranlog entry 3.61.0, you'll find:

Example I.8 Example Employee modifications, output in full mode

```
2.118.0  BeginData session: 5 beginId:(101033 0)
  numObjs:11  pad1:0 pad2:0
```

[details for other objects omitted]

```
objId 38808321 isNewObj=1  class 39017217 segId:9 len 3 gcSz 3 psize
48 bits 0x201 page 1081
Oop values:  38808577  306(sI 38) 38808833

objId 38808577 isNewObj=1  class 38726401 segId:9 len 3 gcSz 3 psize
48 bits 0x201 page 1081
Oop values:  38809089  38809601  20

objId 38808833 isNewObj=1  class 38715649 segId:9 len 5 gcSz 5 psize
64 bits 0x201 page 1081
Oop values:  38809857  20  38810113  38810369  777714(sI 97214)

objId 38809089 isNewObj=1  class 74753 segId:9 len 7 gcSz 0 psize 32
bits 0x280 page 1081
Bytes:  Patrick

objId 38809601 isNewObj=1  class 74753 segId:9 len 5 gcSz 0 psize 32
bits 0x280 page 1081
Bytes:  Ohara

objId 38809857 isNewObj=1  class 74753 segId:9 len 13 gcSz 0 psize 40
bits 0x280 page 1081
Bytes:  2556 Fir Blvd

objId 38810113 isNewObj=1  class 74753 segId:9 len 7 gcSz 0 psize 32
bits 0x280 page 1081
Bytes:  Ashford
```

```
objId 38810369 isNewObj=1  class 74753 segId:9 len 2 gcSz 0 psize 32
bits 0x280 page 1081
Bytes:  OR
```

So, for the Employee object 38808321, we find:

```
objId 38808321 isNewObj=1  class 39017217 segId:9 len 3 gcSz
3 psize 48 bits 0x201 page 1081
Oop values:  38808577  306(sI 38) 38808833
```

Indicating that this is an instance of class 39017217, the Employee class, which has three instance variables: name, age, and address. By position, we can identify the data in the instance variables:

name: 38808577 - the OOP of an instance of Name, found later in the entry
age: 306 - the OOP of the SmallInteger (sI) 38
address: 38808833 - the OOP of a instance of Address, found later in the entry

Looking at the Name object 38808577 we find:

```
objId 38808577 isNewObj=1  class 38726401 segId:9 len 3 gcSz
3 psize 48 bits 0x201 page 1081
Oop values:  38809089  38809601  20
```

Indicating that this is an instance of the class with OOP 38726401 (Name). Name contains three instance variables, last, first, and middle. By position, we see the data is:

last: 38809089 - the OOP of a String, described below
first: 38809601 - the OOP of a String, described below
middle: 20 (the OOP of nil) - in this example, no middle name was set

For the last name object 38809089, we find:

```
objId 38809601 isNewObj=1  class 74753 segId:9 len 5 gcSz 0
psize 32 bits 0x280 page 1081
Bytes:  Ohara
```

Indicating the last name is the string “Ohara”.

By a similar process you can follow the trail of objects to examine the structure of other subobjects in the Name and Address objects.

I.5 Further Analysis

Class Operations

To find all objects created or modified that belong to a particular class, first generate `printlogs` output in full mode of the tranlogs of interest. Each time an object of that class is created or modified, the full tranlog entry includes the line

```
class <OOP>
```

Use the unix `grep` command to find all references to the OOP of the class.

For example, to find all creation or modification of any instance of our example class Employee in the printlogs full output.

Since the Employee class is OOP 39017217, execute the `grep` command:

```
$>  grep "class: 39017217" tranlogfull.txt
```

this will produce output of the form:

```
objId 38804481 isNewObj=1  class 39017217 segId:9 len 3 gcSz
3 psize 48 bits 0x201 page 1087
objId 38808321 isNewObj=1  class 39017217 segId:9 len 3 gcSz
3 psize 48 bits 0x201 page 1081
objId 40062465 isNewObj=1  class 39017217 segId:9 len 3 gcSz
3 psize 48 bits 0x201 page 1317
objId 38808321 class 39017217 segId:9 len 3 gcSz 3 psize 48
bits 0x1 page 871
objId 38808321 class 39017217 segId:9 len 3 gcSz 3 psize 48
bits 0x1 page 900
```

This gives the first line from the entry creating/modifying the object belonging to that class. From this, you can use other commands to search for and/or track the history of these objects.

Deleted Objects

An object-oriented system doesn't actually delete objects; objects cease to be referenced and are eventually garbage-collected. Noting the removal of an object requires examining the references to that object (such as from a collection) and identifying when the referencing object was modified in such a way that the object of interest is no longer referenced. Meanwhile, as you examine the `printlogs` output, you may find references to the OOP of a dereferenced object in garbage collection tranlog entries.

Managing Volume

As noted above, the `printlogs` produce a very large amount of output. GemStone tranlogs may be multiple GB in size. The output of `printlogs` in normal mode will be somewhat larger than the original tranlog (the `printlogs` output, being human readable, is less dense). The output from this script in full mode is much larger.

To manage the volume:

- Avoid configuring your system with very large tranlogs.
- Ensure that you have plenty of disk space available before beginning analysis.
- Print only the tranlogs containing data you need. Use the `searchlogs` script to identify exactly where the required information is located.
- Make sure that only the relevant tranlogs are in the current directory; move the unneeded ones elsewhere. However, you must retain a continuous set of tranlogs without gaps in sequence, and you must include the tranlog with the original log entry, in order to have the UserId and other information provided.
- Once you have printed the output, use the UNIX utility `grep -n` to locate the lines of interest, and the UNIX utility `split -l` to break the resulting file up into more manageable size chunks.

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

T

U

V

W

Z